01

자료구조의 개요

01. 자료구조의 개요

I. 자료구조의 개념

- 자료는 잘 정리되어 있을수록 효과적으로 사용할 수 있음.
- 자료를 효율적으로 표현하고 저장하고 처리하는 것을 '자료구조'라고 함.

(a) 자료구조를 적용하기 전 (b) 자료구조를 적용한 후

그림 6-1 수산물 코너의 진열대에 자료구조 개념을 적용한 예

- **자료(Data) :** 프로그램으로 처리하고자 하는 데이터.
- **자료구조(Data Structure) :** 데이터의 구조적 특성이 잘 살도록 체계적으로 데이터를 저장하고 사용하는 방법.

01. 자료구조의 개요

II. 자료구조의 분류

▪ 단순 구조

- **단순 구조(Simple Structure) :** 기본 자료형에 따라 다시 분류되며, 정수, 실수, 문자, 문자열 등의 자료형이 있음.

▪ 선형 구조

- **선형 구조(Linear Structure) :** 어떤 순서에 따라 데이터들이 한 줄로 늘어선 자료구조로, 배열, 연결 리스트, 스택, 큐 등이 있음.

▪ 비선형 구조

- **비선형 구조(Nonlinear Structure) :** 비순차적인 성질을 지닌 데이터들을 표현한 구조로, 트리와 그래프가 있음.

01. 자료구조의 개요

II. 자료구조의 분류

그림 6-2 자료구조의 분류

01. 자료구조의 개요

III. 자료구조와 알고리즘의 관계

- **알고리즘(Algorithm) :** 적합한 순서대로 수행하기만 하면 제한된 시간 내에 주어진 문제를 해결할 수 있는 명령어들의 집합.
- 프로그램은 컴퓨터에서 실제로 수행되도록 구현된 알고리즘인데, 프로그램을 작성하려면 자료구조가 필요하므로 알고리즘과 자료구조는 서로가 서로에게 영향을 미치는 아주 중요한 개념들임.

III. 자료구조와 알고리즘의 관계

- **라면을 끓여야 하는 상황에 필요한 알고리즘과 자료구조는?**
  - 알고리즘에 적힌 명령들을 순서대로 실행하면 제한된 시간 안에 라면이 완성되므로 준비된 데이터는 알고리즘을 수행하는 데 적합한 자료임.

(a) 데이터 : 라면 재료 (b) 알고리즘 : 라면 레시피

그림 6-3 데이터와 알고리즘의 관계 : 라면 끓이기 준비

III. 자료구조와 알고리즘의 관계

- **라면을 좀 더 효율적이고 편리한 방법으로 끓이려면?**
  - '정해진 시간에 맞게 가장 상단의 재료부터 넣는 구조'로 라면 레시피인 알고리즘을 훨씬 간단하게 수행할 수 있음.

그림 6-4 자료구조를 통해 효율적으로 라면을 끓이는 방법

01. 자료구조의 개요

III. 자료구조와 알고리즘의 관계

- 시스템을 만드는 데 필요한 데이터를 효율적으로 배치하는 것은 **‘자료구조’**
- 시스템을 구현하고 실행 순서 및 방법을 체계적으로 표현하는 것은 **‘알고리즘’**

02

배열

02. 배열

I. 배열의 개념

- **배열(Array) :** 자료형이 같은 데이터를 순서대로 나열한 뒤 메모리에 연속으로 저장해 만든 자료 그룹.

- [그림 6-5]는 트럼프 카드의 클로버 부분을 배열로 나열한 것.
  - 각각의 카드를 변수로 선언하면 13개의 변수를 개별적으로 사용해야 하지만, 하나로 묶어 배열로 만들면 한 번만 선언해도 전체에 대한 변수를 만들 수 있음.
  - 각각의 위상에 해당하는 숫자와 문자가 배열의 요소로 작용해 다루기도 편함.

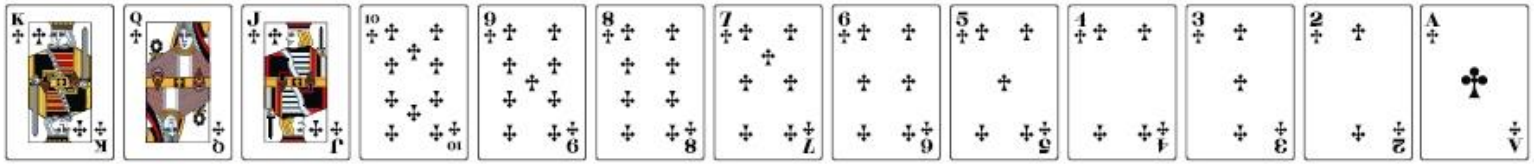


그림 6-5 배열의 실생활 예 : 트럼프 카드

02. 배열

II. 배열의 구조

- **인덱스(Index) :** 배열 요소를 구별할 때 사용하는 번호로, 파이썬을 포함해, 프로그래밍 언어에서 인덱스는 항상 0부터 시작.
- 특정 배열 요소를 사용할 경우 '**배열 이름[배열 요소의 인덱스]**'로 지정하고 각각의 변수처럼 사용.

그림 6-6 colors의 배열

02. 배열

II. 배열의 구조

- [그림 6-7]은 파이썬에서 'colors'라는 변수를 하나 생성해 자료형으로 배열(리스트)을 할당한 경우.
- 배열(리스트)을 할당하기 위해 대괄호를 사용.
- 배열은 특정 위치에 있는 데이터로 빠르게 접근할 수 있음.

  ex) 2번째 colors인 'blue'는 colors[1]이므로 인덱스만 알면 바로 찾을 수 있음.

```
colors = ['red', 'blue', 'green', 'black']
```

(a) colors 배열에 데이터를 배치하는 파이썬 코드

(b) colors 배열의 데이터 배치

그림 6-7 파이썬으로 colors 배열 만들기

02. 배열

II. 배열의 구조

하나 더 알기 **2차원 배열**

- **2차원 배열 :** 인덱스를 2개 사용하는 배열.
- 첫 번째 인덱스는 행을, 두 번째 인덱스는 열을 나타냄.

	1열	2열
1행	Colors[0][0]	Colors[0][1]
2행	Colors[1][0]	Colors[1][1]
3행	Colors[2][0]	Colors[2][1]

그림 6-8 2차원 배열

02. 배열

III. 배열에서의 데이터 추가 및 삭제

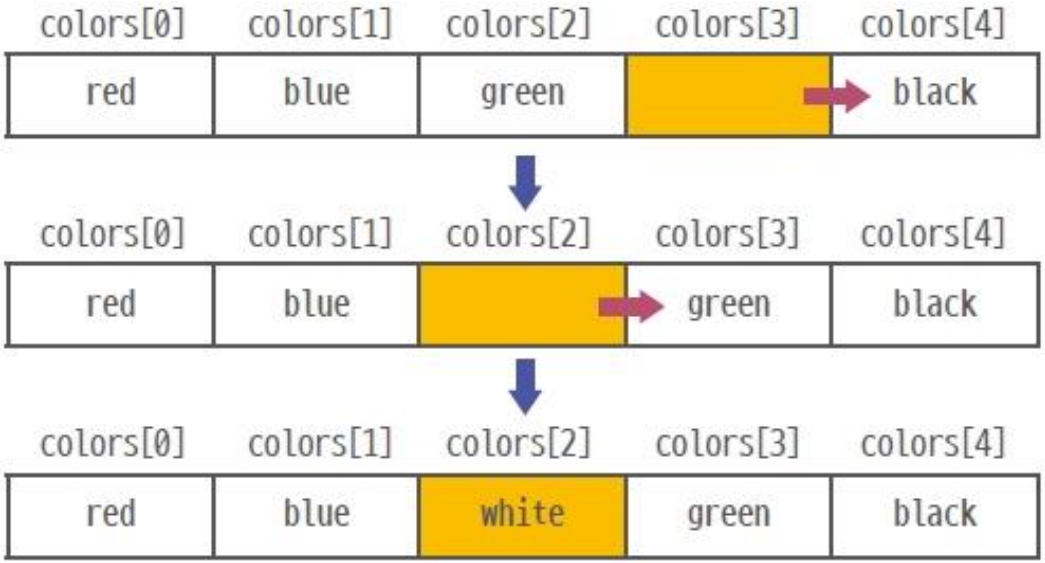


그림 6-9 배열에 데이터 'white' 삽입하기

02. 배열

III. 배열에서의 데이터 추가 및 삭제

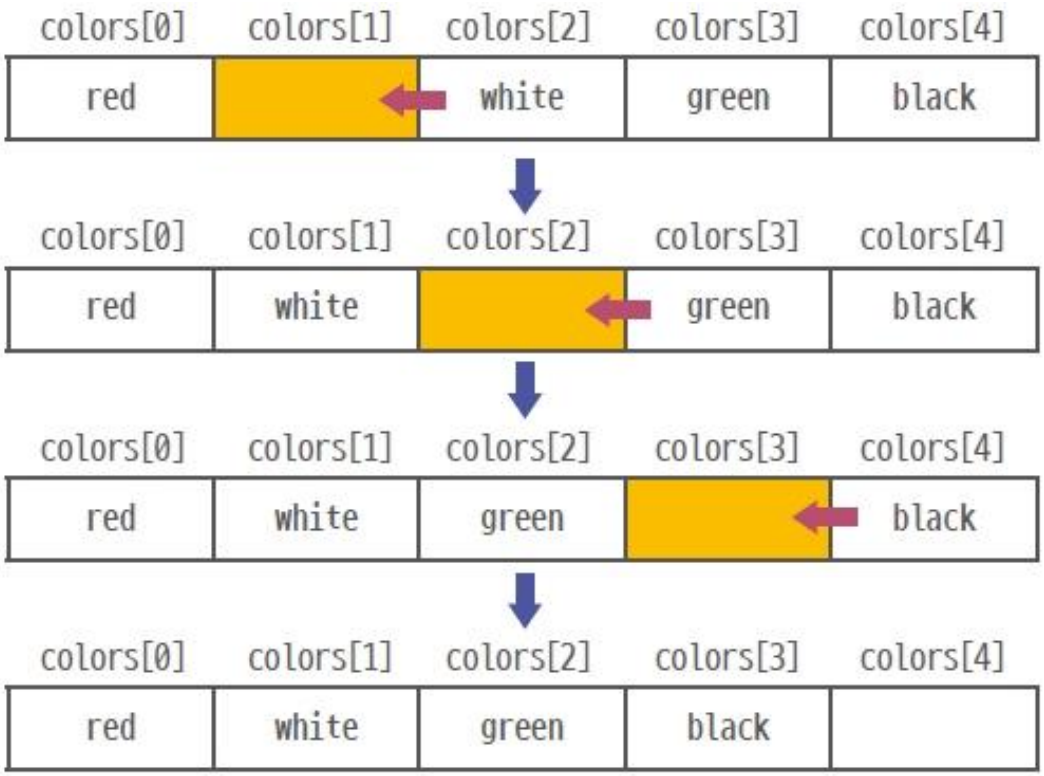


그림 6-10 배열에서 데이터 'blue' 삭제하기

03
연결 리스트

03. 연결 리스트

I. 연결 리스트의 개념

- 배열은 논리적인 순서와 물리적인 순서가 같아서 데이터의 위치를 찾기는 쉽지만 추가나 삭제를 한 후에는 연속적인 물리적 위치를 유지하기 위해 데이터 자리를 추가적으로 옮겨야 함. 그래서 작업시간이 많이 필요하고, 메모리 사용이 비효율적임. 이러한 문제를 개선한 데이터 표현 방법이 연결 리스트임

- **연결 리스트(Linked List) :** 연속된 물리적 주소에 따라 데이터 순서를 표현하는 것이 아니라, 각 데이터에 저장된 다음 데이터의 주소(링크)에 따라 순서가 연결되는 방식.
- 데이터의 논리적인 순서와 물리적인 순서가 일치하지 않아도 됨.

03. 연결 리스트

II. 연결 리스트의 구조

- **노드(Node) :** 연결 리스트의 데이터는 <원소, 주소> 단위로 구성되어 있으며 다음에 이어질 데이터에 대한 주소를 포함하는데, 이를 합쳐서 노드라고 함.
- **노드의 구조 :** 원소 값을 저장하는 **데이터 필드와** 다음 노드 주소를 저장하는 **링크 필드로** 되어 있음

그림 6-11 노드의 구조

- **연결 리스트의 예**
  - 각각의 노드는 링크로 연결되어 있음.

그림 6-12 연결 리스트의 예

더이상 연결될 것이 없다

03. 연결 리스트

III. 연결 리스트의 분류

▪ 단순 연결 리스트

- **단순 연결 리스트(Singly Linked List) :** 노드의 링크 필드가 하나이며, 링크 필드는 다음 노드와 연결되는 가장 기본 구조.

▪ 원형 연결 리스트

- **원형 연결 리스트(Circular Liked List) :** 단순 연결 리스트의 마지막 노드가 리스트의 첫 번째 노드를 가리키게 하여 리스트 구조를 원형으로 만든 구조.

▪ 이중 연결 리스트

- **이중 연결 리스트(Doubly Linked List) :** 양쪽으로 모두 순회할 수 있도록 노드를 연결해 단순 연결 리스트와 원형 연결 리스트의 단점을 보완한 구조.

03. 연결 리스트

IV. 연결 리스트에서의 데이터 추가 및 삭제

- 연결 리스트에 데이터를 추가하는 방법

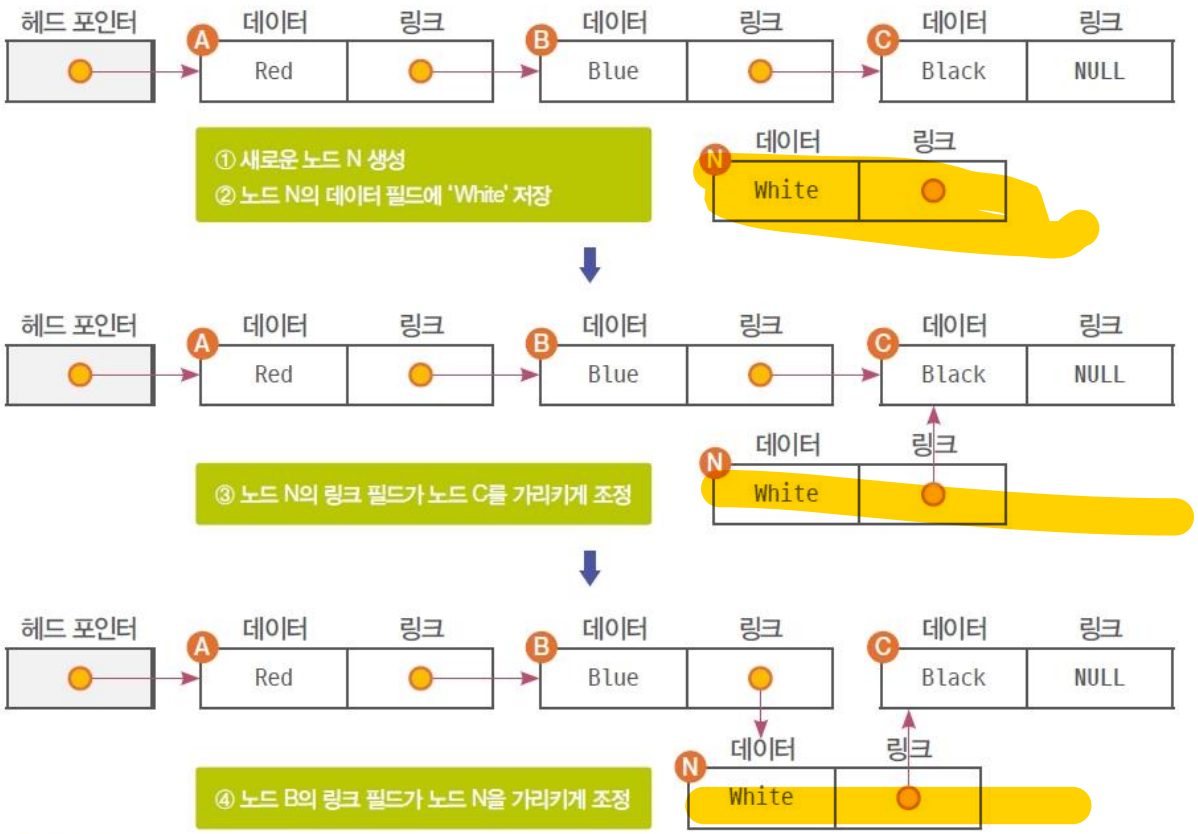


그림 6-13 연결 리스트에서의 데이터 추가

03. 연결 리스트

IV. 연결 리스트에서의 데이터 추가 및 삭제

- 연결 리스트에 데이터를 삭제하는 방법

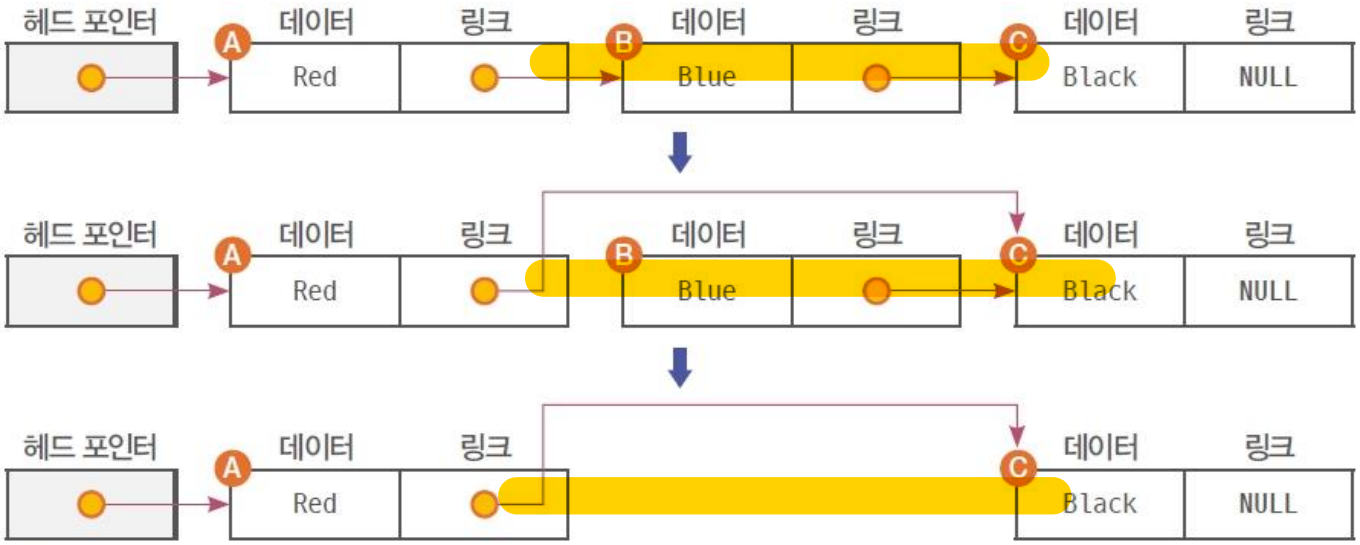


그림 6-14 연결 리스트에서의 데이터 삭제

03. 연결 리스트

IV. 연결 리스트에서의 데이터 추가 및 삭제

하나 더 알기 **배열과 연결 리스트 비교**

구분	배열	연결 리스트
데이터 검색 기능	인덱스를 알면 위치와 상관없이 데이터를 쉽게 찾을 수 있음.	첫 노드부터 순서를 거쳐 찾아가야 함.
메모리 공간	자료를 위한 메모리 공간만 필요.	자료를 위한 메모리 공간 외에도 링크 필드의 포인터를 위한 메모리 공간이 추가로 필요.
메모리 사용	정적인 메모리를 사용하기 때문에 작업 전 최대 자료 개수를 미리 알고 있어야 함.	프로그램 실행 중에도 노드 단위로 메모리를 할당받을 수 있어 공간을 보다 효율적으로 사용할 수 있음.
데이터 추가 및 삭제	복잡하고 어려움.	편하고 쉬움.

04

스택

04. 스택

I. 스택의 개념

- **스택(Stack) :** 데이터를 차곡차곡 쌓아 올린 형태로 구성된 자료구조.

그림 6-15 스택의 실생활 예 : 쌓인 책과 이삿짐 박스

쌓고 줄 세우는 데이터, 스택과 큐(04:38)

04. 스택

II. 스택의 구조

- 스택에서는 톱(Top)으로 정해진 맨 위에서만 데이터의 추가, 삭제 작업이 가능하며, 구조의 중간에서는 데이터를 추가하거나 삭제하지 못함.
- **LIFO(Last In First Out) :** 데이터를 삭제할 때도 가장 마지막에 삽입(Last In)된 데이터가 가장 먼저 삭제(First Out)된다는 구조.

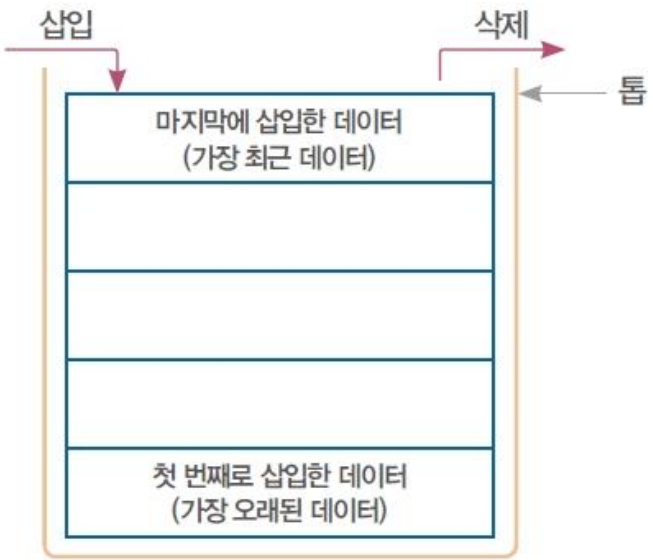


그림 6-16 스택의 구조

04. 스택

III. 스택에서의 데이터 추가 및 삭제

- **푸시(Push) :** 스택에서 데이터를 추가(저장)하는 작업.
- **팝(Pop) :** 데이터를 삭제(추출)하는 작업.

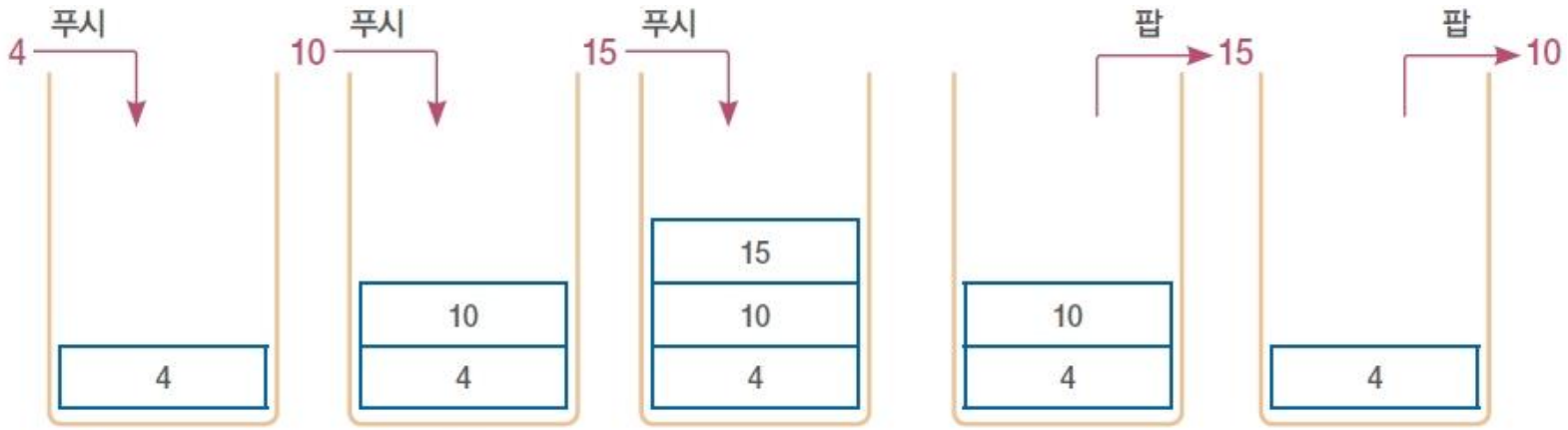


그림 6-17 스택에서의 데이터 추가(푸시) 및 삭제(팝)

04. 스택

IV. 파이썬에서의 스택 표현

- 파이썬에서는 리스트를 사용해 스택 구현 가능.
- 스택과 관련된 함수(Method)로는 append( )와 pop( )이 있음
  - append( ) 함수 : 리스트 끝에 새 항목을 추가하는 함수.
  - pop( ) 함수 : 리스트 끝 항목을 삭제하는 함수.

코드 6-1 파이썬에서의 스택 구현

```
>>> a = [1, 3, 5, 7, 9]
>>> a.append(15)
>>> a
[1, 3, 5, 7, 9, 15]
>>> a.append(20)
>>> a
[1, 3, 5, 7, 9, 15, 20]
>>> a.pop()
20
>>> a.pop()
15
```

그래프와 트리의 차이는 순환이 있는지 없는지 확인 해야 한다

05

큐

05. 큐

I. 큐의 개념

- **큐(Queue) :** 데이터가 한 방향에서만 삽입되고 반대 방향으로만 삭제되는 자료구조로, 스택의 반대 개념.

그림 6-18 큐의 실생활 예 : ATM 대기자

05. 큐

II. 큐의 구조

- 한쪽 끝을 프런트(Front)로 정해 삭제 연산만 수행하고, 다른 쪽 끝을 리어(Rear)로 정해 삽입 연산만 수행.
- **FIFO(Fist In First Out) :** 먼저 삽입(First in) 된 데이터가 먼저 삭제(First out) 되는 구조.

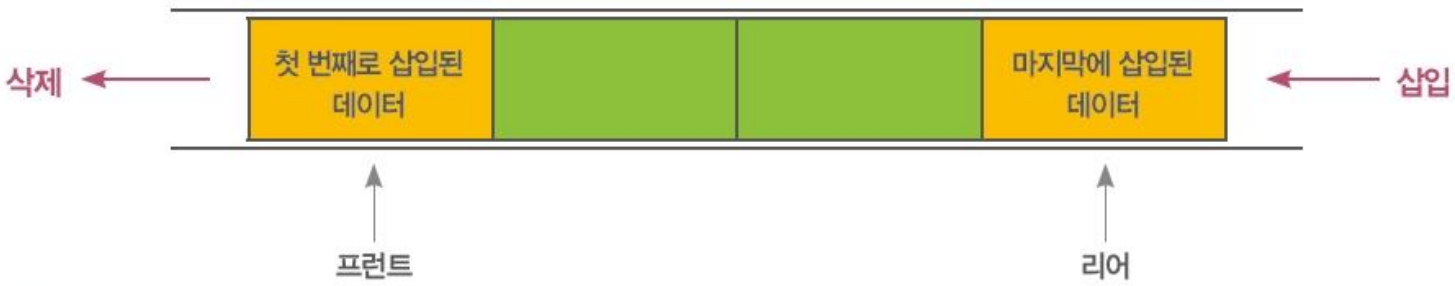


그림 6-19 큐의 구조

III. 큐에서의 데이터 추가 및 삭제

- 큐에서 데이터 추가 및 삭제

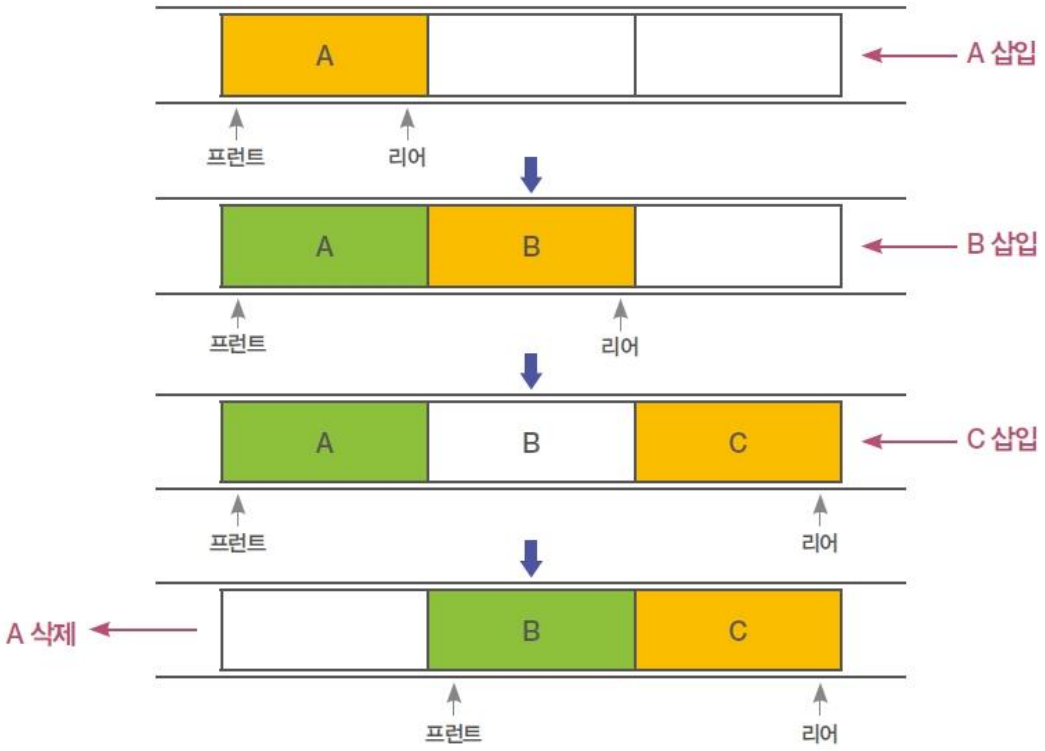


그림 6-20 큐에서의 데이터 추가 및 삭제

05. 큐

IV. 파이썬에서의 큐 표현

- 기본 구조는 스택 구현 방법과 같음.
- pop( ) 함수는 리스트의 마지막 값을 삭제하는 함수이므로, 인덱스가 0번째인 값을 추출한다는 의미로 pop(0)을 사용하면 됨. pop(0)은 맨 처음 값을 삭제함을 의미.

코드 6-2 파이썬에서의 큐 구현

```
>>> a = [1, 2, 3, 4, 5]
>>> a.append(10)                # a = [1, 2, 3, 4, 5, 10]
>>> a.append(20)                # a = [1, 2, 3, 4, 5, 10, 20]
>>> a.pop(0)
1
>>> a.pop(0)
2
```

06
비선형 자료구조

06. 비선형 자료구조

- **비선형 자료구조(Non-linear Data Structure)** : 데이터 간의 상관관계를 연결 상태로 표현해야 할 때 사용하는 구조.
- **비선형 자료구조의 종류 :** 트리와 그래프

06. 비선형 자료구조

I. 트리

- **트리(Tree) :** 나무를 뒤집어 놓은 모습의 자료구조로, 데이터들이 1 : n 관계인 비선형 구조로 놓여 있음.
- 트리는 노드와 간선의 집합으로 이루어져 있음.
  - 노드(Node) : 트리의 원
  - 간선(Edge) : 노드와 노드를 연결한 선
- 트리는 계층 관계를 가지는데, 트리의 노드를 이용해 부모-자식 간 관계를 표현할 수 있음.
  - 루트 노드 : 트리 맨 위에 위치한 노드
  - 단말 노드 : 가장 아래에 위치한 노드들

06. 비선형 자료구조

I. 트리

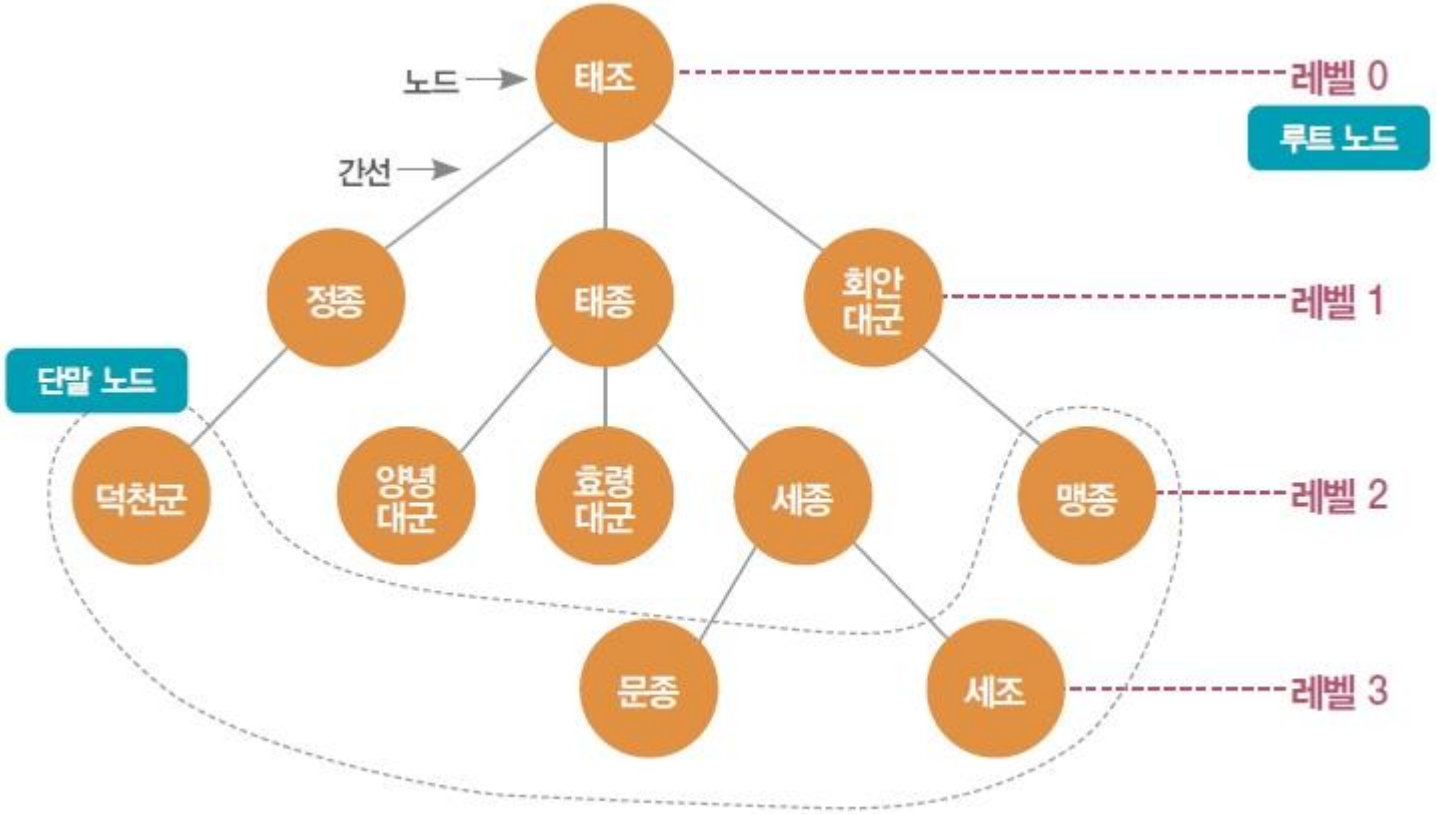


그림 6-21 트리의 실생활 예 : 가계도

06. 비선형 자료구조

I. 트리

- **이진 트리(Binary Tree) :** 노드의 자식 노드를 2개 이하로 정해 놓은 구조로, 왼쪽과 오른쪽, 2개의 자식 노드만 가질 수 있음.

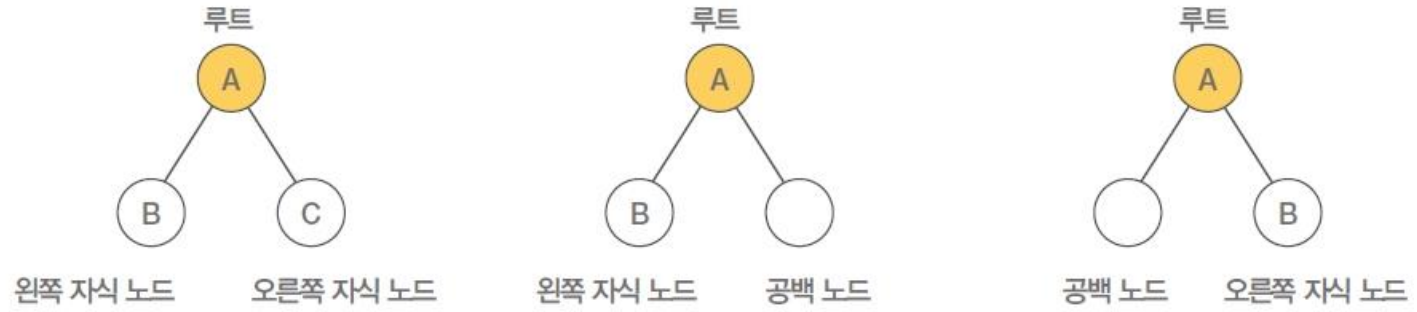


그림 6-22 이진 트리의 기본 구조

06. 비선형 자료구조

I. 트리

- [그림 6-23]은 루트 노드 A의 왼쪽 자식 노드 B를 루트 노드로 하는 서브 트리와 오른쪽 자식 노드 C를 루트 노드로 하는 서브 트리로 구성된 이진 트리임.

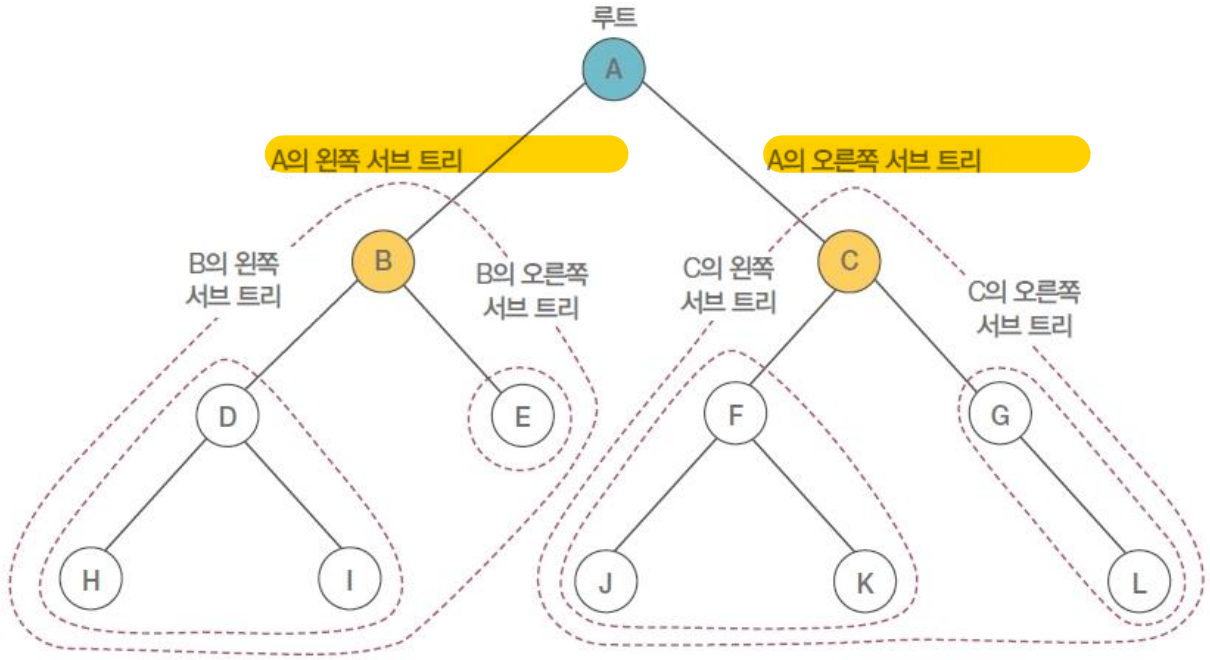


그림 6-23 서브 트리가 합쳐진 이진 트리

06. 비선형 자료구조

II. 그래프

- **그래프(Graph) :** 연결된 데이터들의 n : n(다 : 다) 관계를 표현하는 자료구조.

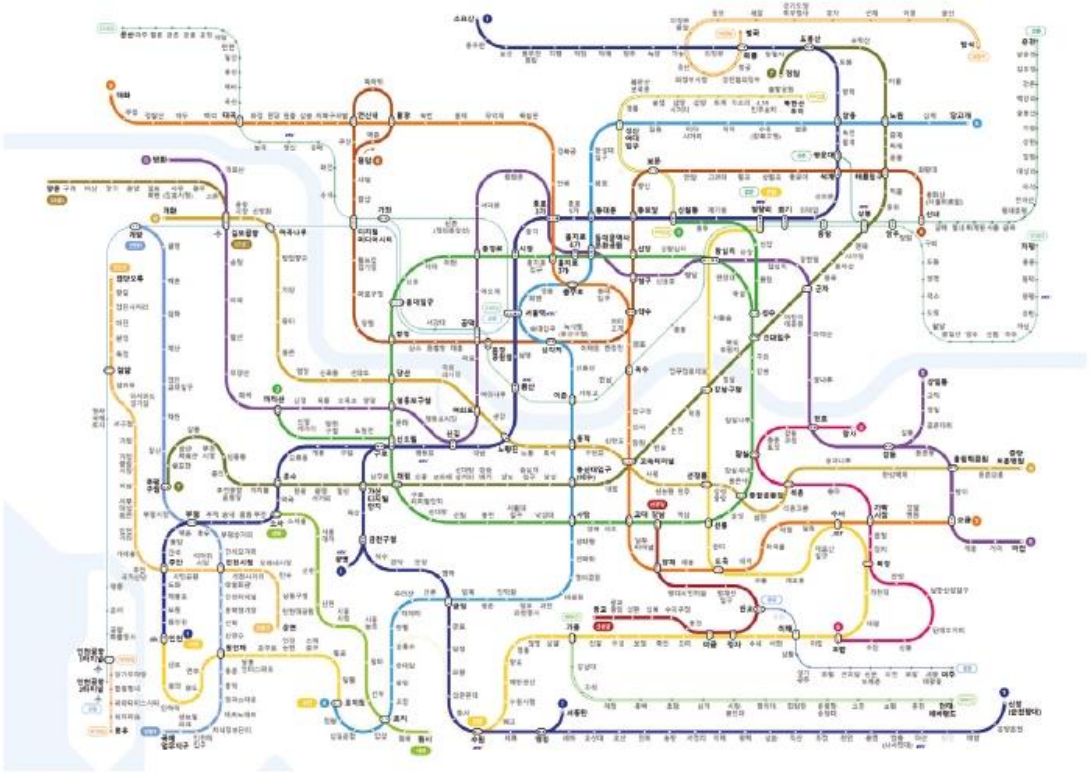


그림 6-24 그래프의 실생활 예 : 지하철 노선도

06. 비선형 자료구조

II. 그래프

- 그래프는 연결할 객체를 나타내는 정점과 객체를 연결하는 간선의 집합.
- 그래프와 트리의 차이점은 그래프에는 사이클이 존재한다는 것.
- 그래프(G)는 G=(V,E)로 정의. V는 그래프에 있는 정점의 집합을, E는 정점을 연결하는 간선의 집합을 뜻함.

06. 비선형 자료구조

II. 그래프

- **그래프의 분류** 그래프와 트리의 차이는 순환이 있는지 없는지 확인 해야 한다
  - 무방향 그래프 : 두 정점을 연결하는 간선의 방향이 정해져 있지 않은 그래프.
  - 방향 그래프 : 간선에 방향이 정해져 있는 그래프.

(a) 무방향 그래프 구조

(b) 방향 그래프 구조

그림 6-25 무방향 그래프와 방향 그래프 구조

II. 그래프

- 방향 그래프는 실생활에 적용 중인 구조이기도 함.
- 숫자로 표시된 노드는 도시 이름, 간선의 숫자는 도시 간 거리라고 가정하면 도시 간 최단 거리를 구하는 것이 내비게이션이 구현하고 있는 알고리즘임.

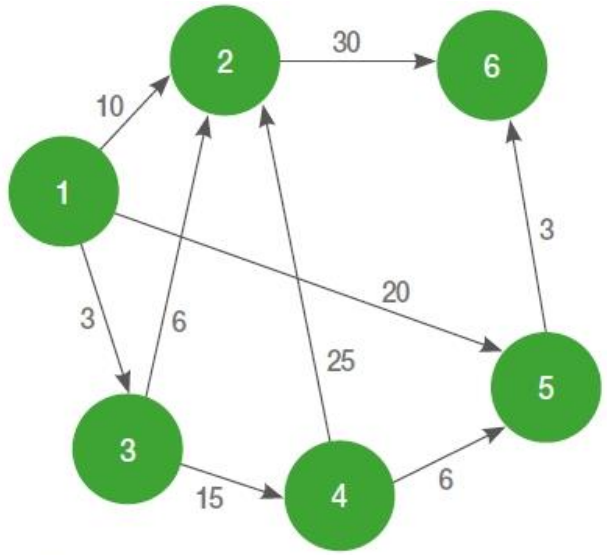
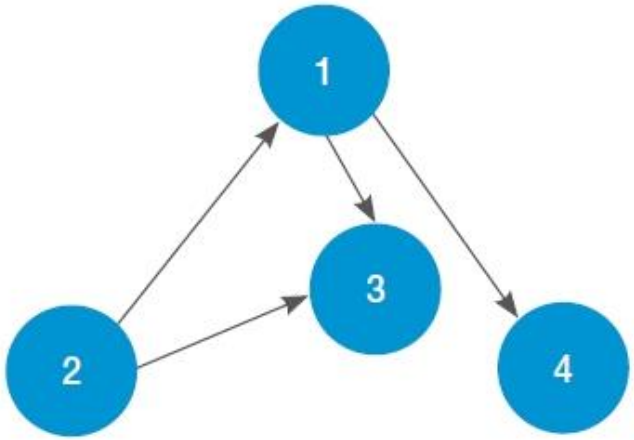


그림 6-26 방향 그래프의 실생활 예 : 내비게이션 알고리즘

II. 그래프

- [그림 6-27] (a)의 방향 그래프는 [그림 6-27] (b)처럼 2차원 배열로 표현할 수 있음.
- 예를 들어, 2행 3열의 값이 1이라는 의미는 노드 2에서 노드 3으로 향하는 간선이 존재한다는 의미로 해석 가능.

그래프를 2차원 배열로 표현이 가능한다
– 접근 여부 확인 (0,1)



(a) 그래프

	1	2	3	4
1	0	0	1	1
2	1	0	1	0
3	0	0	0	0
4	0	0	0	0

(b) 2차원 배열

그림 6–27 그래프를 2차원 배열로 구현