제 1 장

자료구조/알고리즘 소개

1) 자료구조란?
2) 알고리즘이란?
3) 실습환경 구성

01. 자료구조란?

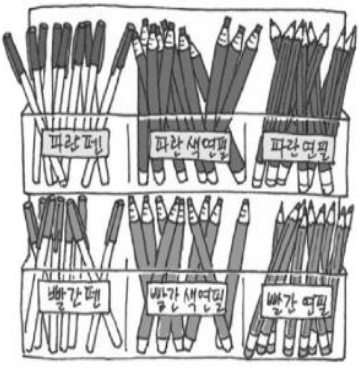
자료구조

자료구조(Data Structure)는 데이터를 효율적으로 저장, 관리, 조작할 수 있도록 하는 효율적인 데이터의 형태를 의미하며, 컴퓨터 과학과 소프트웨어 개발에서 매우 중요한 역할을 한다.

자료구조의 중요성

자료구조는 데이터 검색, 삽입, 삭제와 같은 연산을 효율적으로 수행할 수 있도록 데이터를 체계적으로 관리하는 방법을 제공한다.
또한, 자료구조는 메모리 사용을 최적화하는 데에 중요한 역할을 할 수 있다. 한가지 예로, 연결 리스트(Linked List)는 동적 메모리 할당을 통해 필요할 때만 메모리를 사용하여 메모리 낭비를 줄이는 등 메모리 사용을 최적화하는데 도움이 될 수 있다.

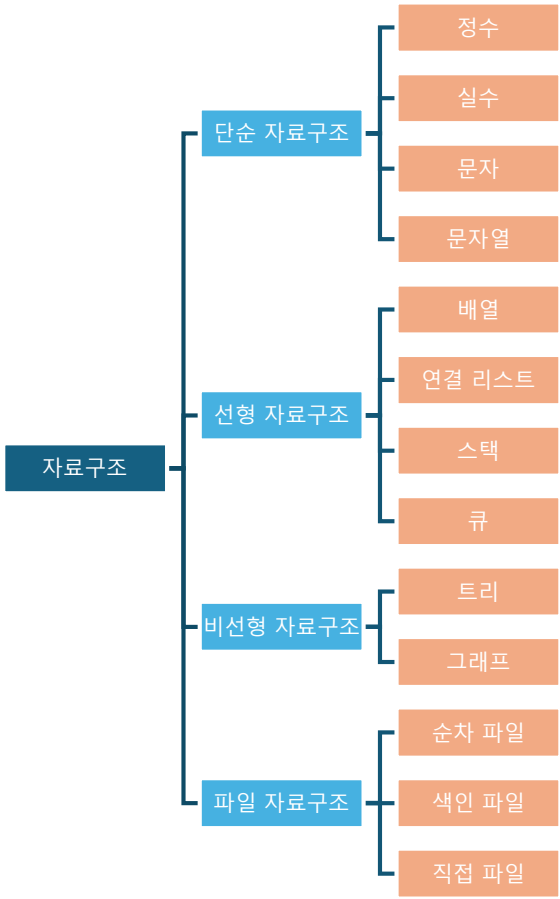
자료구조가 없는 연필 꽂이



자료구조가 있는 연필 꽂이

자료구조의 종류

자료구조는 크게 단순 자료구조, 선형 자료구조, 비선형 자료구조, 파일 자료구조로 분류할 수 있다.



자료구조의 종류

단순 자료구조

단순 자료구조(Simple Data Structure)는 프로그래밍 언어에서 제공하는 데이터 형식을 의미하며, 정수, 실수, 문자, 문자열을 포함한다.

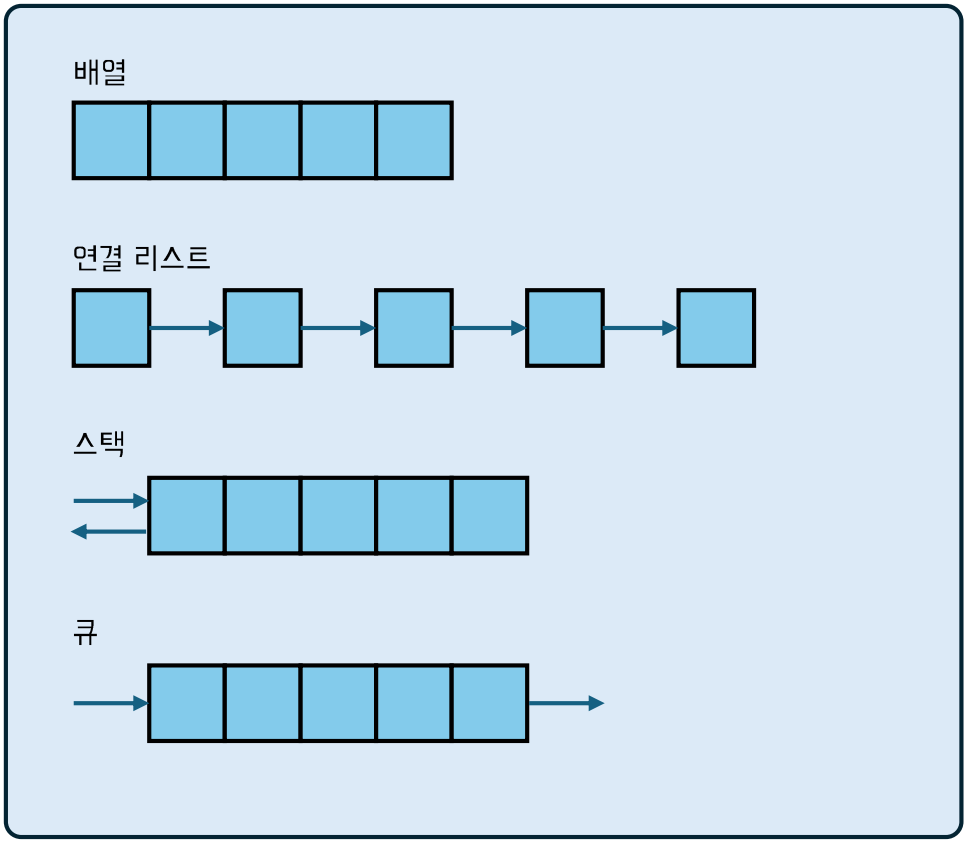
정수는 int로 표현하며 0, 2024, -10 등 소수점이 없는 수를 의미하고, 실수는 float으로 표현하며 0.1, 3.14, 2.123 등 소수점이 있는 수를 의미한다. 문자는 char로 표현하며 '가', 'A', '5' 등 한 글자를 의미하고 작은 따옴표(' ')로 묶어 사용한다. 마지막으로 문자열은 string으로 표현하며 "자료구조", "ABC", "12345" 등 여러 개의 글자를 의미하고 큰 따옴표(" ")로 묶어 사용한다.

정수	0, 2024, -10 등
실수	0.1, 3.14, 2.123 등
문자	'가', 'A', '5' 등
문자열	"자료구조", "ABC", "12345" 등

단순 자료구조의 종류

선형 자료구조

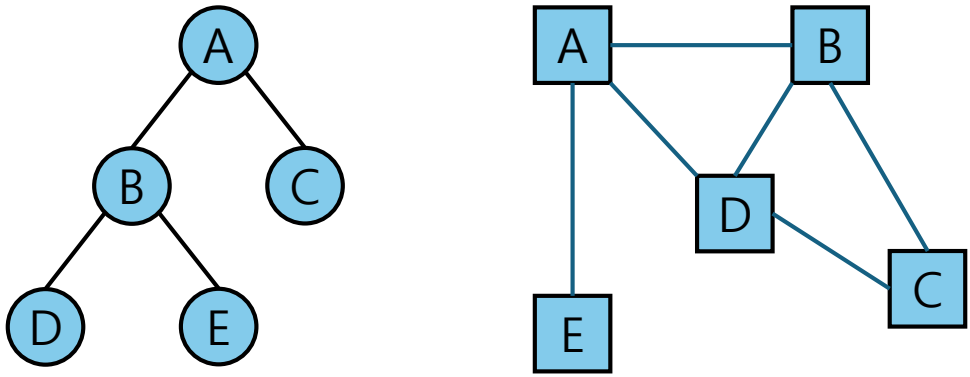
선형 자료구조(Linear Data Structure)는 데이터가 순차적으로 배열된 형태의 자료구조로, 각 데이터 요소가 다음 요소와 일직선상으로 연결되어 있는 형태를 가진다. 데이터의 삽입, 삭제, 검색 등의 접근 방식이 단순하며, 데이터를 순차적으로 처리할 때 유용한 자료구조로서 배열, 연결 리스트, 스택, 큐 등을 포함한다.



선형 자료구조의 종류

비선형 자료구조

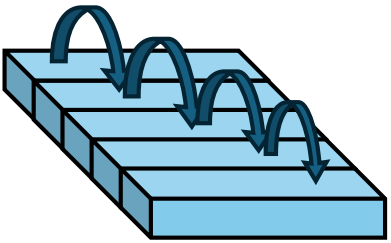
비선형 자료구조(Nonlinear Data Structure)는 데이터 요소가 일직선으로 배열되지 않고, 계층적이거나 네트워크와 같은 복잡한 관계를 가지는 자료구조이다. 이러한 구조는 데이터 간의 관계를 더 유연하게 표현할 수 있도록 하며, 데이터 탐색, 계층 구조 표현 등 복잡한 연결 관계를 가지는 데이터를 처리하는데 유용한 자료구조로서 트리와 그래프가 해당된다.



비선형 자료구조의 종류

파일 자료구조

파일 자료구조(File Data Structure)는 파일 시스템에서 데이터를 효율적으로 저장하고 관리하기 위해 사용하는 자료구조이다.
이러한 자료구조는 데이터를 파일로 조직하고, 파일 시스템의 성능과 효율성을 높이기 위한 다양한 방법들을 포함하며, 파일 내용이 디스크에 저장되는 방식으로 구분되어, 순차 파일, 색인 파일, 직접 파일을 포함한다.



파일 자료구조(순차 파일)

02. 알고리즘이란?

알고리즘

알고리즘(Algorithm)은 어떠한 문제를 해결하기 위한 논리적인 과정을 공식화한 형태로 표현한 것이다.

알고리즘의 특징은 다음과 같다.

- 유한성(Finiteness)

  → 알고리즘이 무한 루프에 빠지지 않고, 명확하게 끝나는 지점이 있어야 함

- 명확성(Definiteness)

  → 알고리즘의 각 단계가 무엇을 수행하는지 명확하고 모호함이 없어야 함

- 입력(Input)

  → 알고리즘은 외부에서 주어진 0개 이상의 입력 값이 있어야 함

- 출력(Output)

  → 알고리즘은 적어도 1개 이상의 출력 값을 생성해야 함

- 효과성(Effectiveness)

  → 알고리즘의 모든 단계는 이론적으로 계산 가능해야 함

알고리즘 표현방법

알고리즘을 표현하는 방법은 크게 4가지로 나눌 수 있다.

- 자연어(Natural Language)

  → 사람이 의사소통을 할 때 사용되는 말로, 상대방이 이해하기 쉽게 표현할 수 있다는 장점이 있다. 하지만 이를 바탕으로 코드를 작성하기 어렵다는 단점이 있다.

- 프로그래밍 언어(Programming Language)

  → 사람이 사용하기 쉬운 문자로 명령을 입력하면 기계어로 바꿔 컴퓨터가 알아들을 수 있도록 만들어진 언어이다. Python, C, JAVA 등이 있다.

- 의사 코드(Pseudo Code)

  → 프로그래밍 언어와 자연어의 중간 형태로, 프로그래밍 언어로 직접 코딩하지 않고, 다른 개발자나 이용자에게 작동 원리를 설명할 때 사용된다.

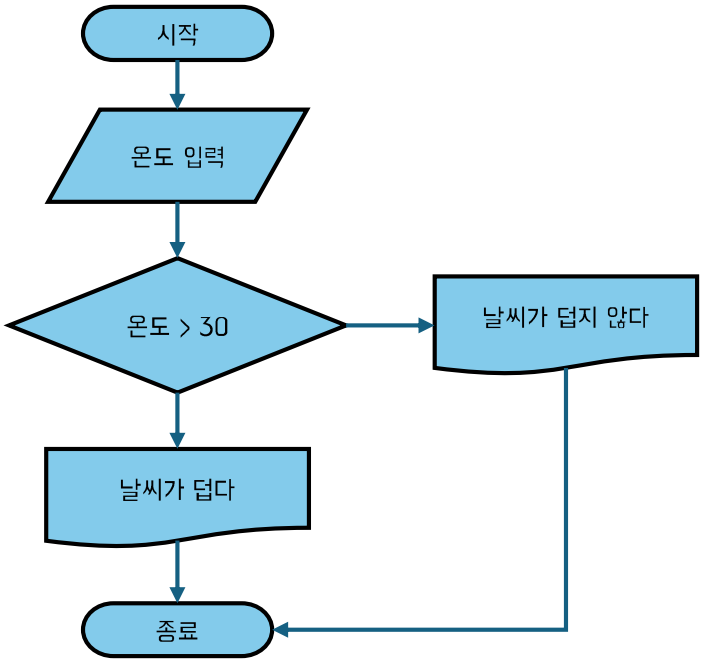
- 순서도(Flow Chart)

  → 프로그램이나 작업의 진행 흐름을 순서에 따라 여러 종류의 상자, 화살표, 문자로 나타내는 방식이다. 간단한 알고리즘은 쉽게 표현할 수 있지만 알고리즘이 복잡해지면 표현하기 어렵다는 단점이 있다.

알고리즘의 표현방법

1-1. 의사 코드 표현 예시

```
1  온도 입력
2  If 온도 > 30
3          "날씨가 덥다" 출력
4  Else
5          "날씨가 덥지 않다" 출력
6  Endif
```



순서도 표현 예시

알고리즘의 성능

알고리즘의 성능은 running time을 이야기 하는 것이 아니다.
Running time은 프로그램을 수행하는 기기에 따라 달라지기 때문에 알고리즘의 성능을 분석하는데 적합하지 않다. 따라서, 알고리즘의 성능을 분석할 때는 공간 복잡도(Space Complexity)와 시간 복잡도(Time Complexity)를 사용한다.

알고리즘의 복잡도 분석

알고리즘의 복잡도 분석은 알고리즘을 구현하지 않고 모든 입력을 고려할 수 있다. 또한, 하드웨어나 소프트웨어 환경과 관계 없이 알고리즘을 분석할 수 있다는 장점이 있다.

1. 시간 복잡도
   → 알고리즘을 수행하는데 걸리는 시간
2. 공간 복잡도
   → 알고리즘을 수행하는데 사용되는 메모리 용량

대부분의 사용자는 메모리를 얼마나 사용하는지 보다 결과가 얼마나 더 빨리 도출되는지에 관심이 있기 때문에 시간 복잡도에 초점을 맞추어 설명을 진행한다.

알고리즘의 복잡도 분석

입력되는 정수를 n이라 하고 이에 따른 시간 복잡도 함수를 $T(n)$이라고 할 때,

$n \times n$을 계산하는 두가지 알고리즘을 고려해보도록 하자.

1-2. 알고리즘 A

```
1  sum = 0
2  for i in range(1,n+1):
3      sum = sum + n
```

1-3. 알고리즘 B

```
1  sum = 0
2  for i in range(1,n+1):
3      for j in range(1,n+1):
4          sum = sum+1
```

	알고리즘 A	알고리즘 B
대입 연산	$n+1$	$n \times n + 1$
덧셈 연산	n	$n \times n$
$T(n)$	$2n+1$	$2n^2+1$

알고리즘 복잡도 분석

알고리즘의 성능 표현방법

n의 크기가 커지면 커질수록 시간 복잡도 함수는 최고차 항에 점근 하는 성질을 가진다. 따라서, 일반적인 상황에서는 최고차 항만 보면 알고리즘의 효율성을 대략적으로 판단할 수 있다. 이러한 특징을 수식으로 나타내기 위해 빅-오 표기법(Big-O notation)을 사용한다.

빅-오 표기법

빅-오 표기법은 시간 복잡도에 미미한 영향을 주는 값들은 무시한다.

1. 상수항 무시
   → 알고리즘이 $O(N+3)$의 복잡도를 가진다면 $O(N)$로 표기
2. 입력 크기가 클 때 계수 무시
   → 알고리즘이 $O(2N)$의 복잡도를 가진다면 $O(N)$로 표기
3. 가장 큰 영향력이 있는 항(최고 차 항)만 표기
   → 알고리즘이 $O(N^2+4N+2)$의 복잡도를 가진다면 $O(N^2)$로 표기

$$O(1) < O(\log N) < O(N) < O(N \log N) < O(N^2) < O(2^n)$$

효율 하락

빅-오 표기법

1. $O(1)$, 상수

   → 입력 데이터의 크기, 데이터의 양에 상관없이 일정한 시간이 소요

   ex) stack, push, pop

2. $O(\log N)$, 로그

   → 입력 데이터가 커질수록 처리 시간이 로그만큼 증가

   데이터가 10배가 되면 처리 시간은 2배

   ex) 이진 트리

3. $O(N)$, 선형

   → 입력 데이터의 크기에 비례해 처리 시간 증가

   데이터가 10배가 되면 처리 시간도 10배

   ex) for 문

4. $O(N\ \log N)$, 선형 로그

   → 입력 데이터가 많아질 수록 처리 시간이 로그 배만큼 증가

   데이터가 10배가 되면 처리 시간은 20배

   ex) 퀵 정렬, 합병 정렬, 힙 정렬

5. $O(N^2)$, 다항

   → 입력 데이터가 많아질 수록 처리 시간이 급수적으로 증가

   데이터가 10배가 되면 처리 시간은 최대 100배

   ex) 삽입 정렬, 버블 정렬, 선택 정렬

6. $O(2^N)$, 지수

   → 입력 데이터가 많아질 수록 처리 시간이 기하급수적으로 증가

   ex) 피보나치 수열

03. 실습환경 구성

파이썬

파이썬(Python)은 1991년 소프트웨어 엔지니어인 '귀도 반 로섬'이 발표한 고급 프로그래밍 언어로, 인터프리터를 사용하는 객체지향 언어이자 동적 타이핑 대화형 언어이다. 파이썬의 사전적 의미는 비단뱀으로, 뱀 두마리가 얽혀있는 모양의 로고를 사용한다.

파이썬은 사용이 쉽고 문법이 간결하기 때문에 빠르게 개발이 가능하며, 복잡한 구문으로 인한 오류 발생을 줄일 수 있다. 또한 다른 언어나 라이브러리에 쉽게 접근하고 연동할 수 있으며, 고성능 어플리케이션이 필요한 경우 C/C++ 과 결합해 사용할 수도 있다.

파이썬의 역사

파이썬

1980년대 말 고안되어 귀도 반 로섬이 1989년 12월 구현을 시작하였다. SETL에서 영감을 받은 ABC 언어의 후속 프로그래밍 언어로서 예외 처리가 가능하고, 아메바 운영체제와 통신이 가능하다.

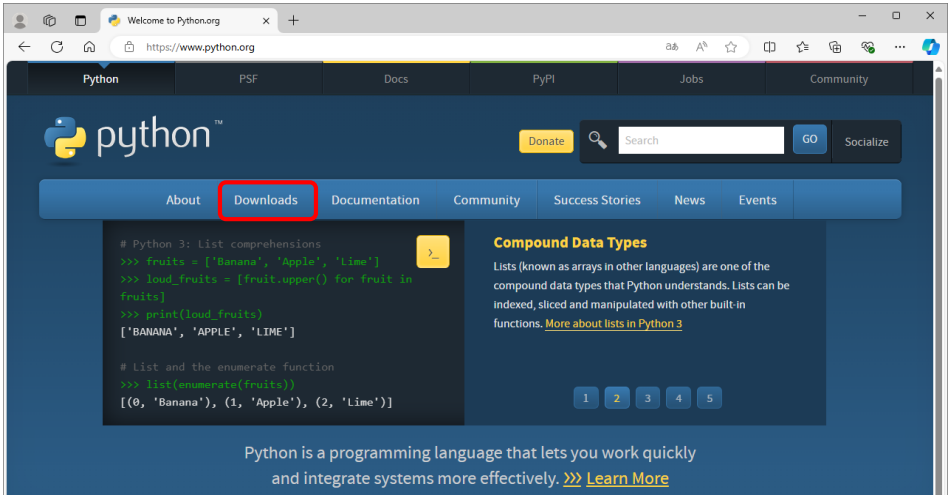
파이썬 2.0

2000년 10월 16일 이후 출시되었으며 메모리 관리를 위한 사이클 감지 Garbage Collector와 유니코드 지원을 포함한 새로운 기능들이 포함되었다.
2020년 1월 1일부로 해당 버전의 파이썬은 지원이 종료되었다.

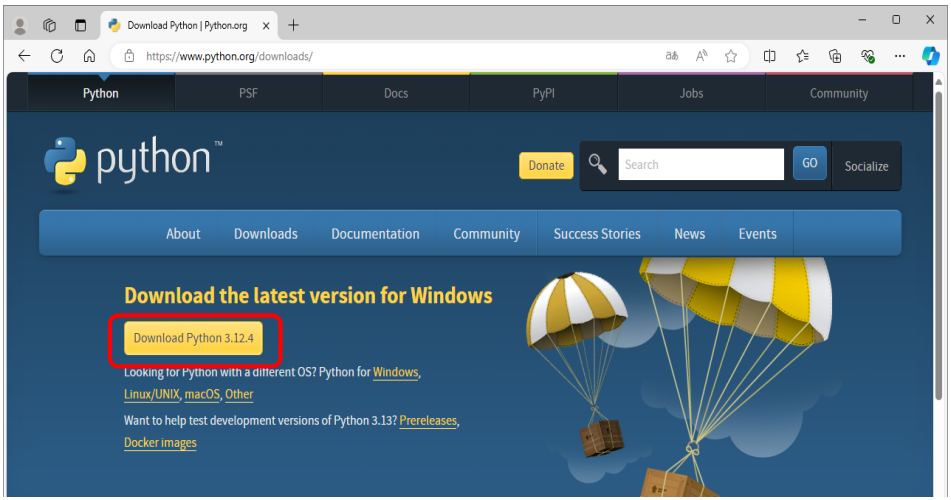
파이썬 3.0

2008년 12월 3일자로 발표되었다. 2.x대 버전의 파이썬과 하위호환성이 없다는 것이 큰 특징이다. 처음 배우는 프로그래머들은 파이썬 3으로 시작하는 것이 권장된다.

파이썬 설치

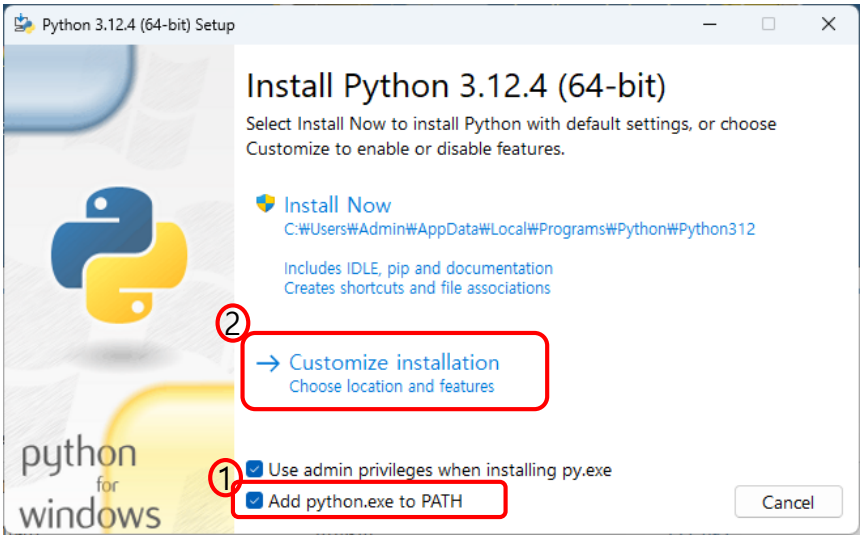


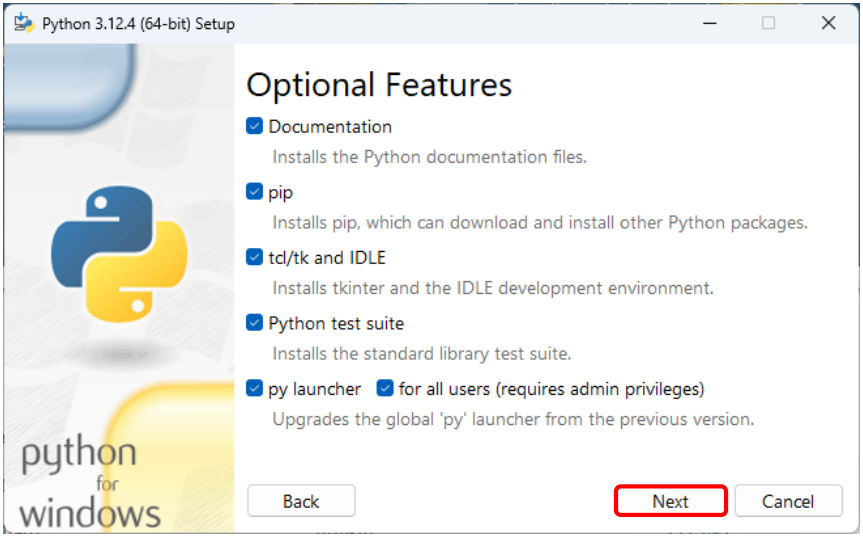
웹 브라우저에서 [https://www.python.org/]에 접속한 뒤, Downloads를 클릭하여 최신 버전을 다운로드 한다.



파이썬 설치

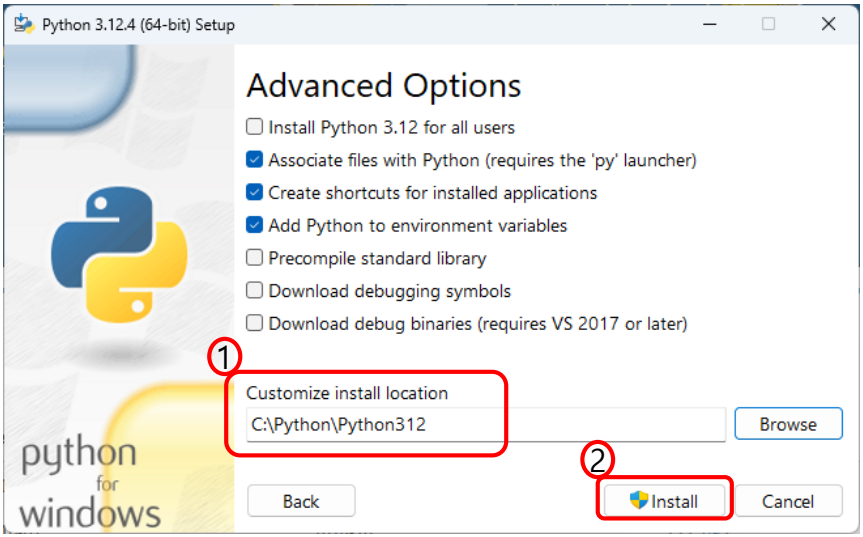
다운로드한 설치 파일 python-3.12.4.exe를 실행하고 [Add python.exe to PATH]를 체크해준 뒤 [Customize installation]을 클릭한다.

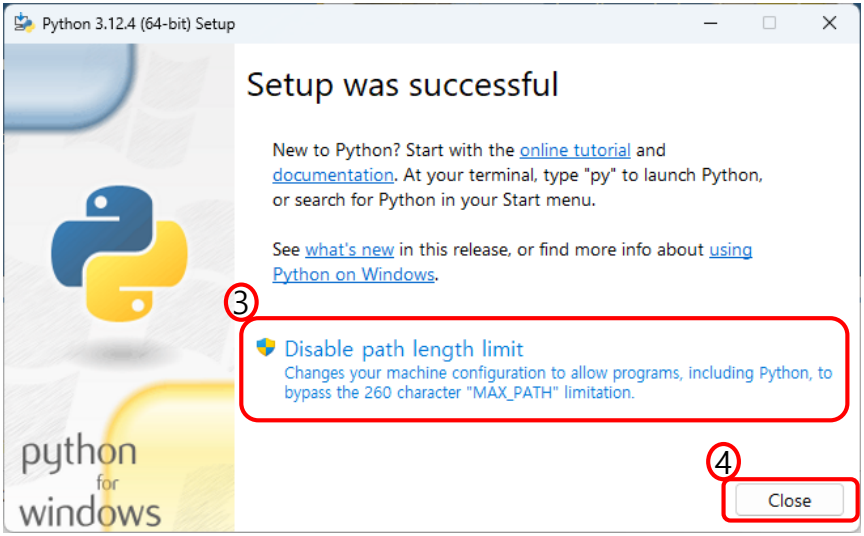




파이썬 설치

[Advanced Options]에서 [Customize install location]을
'C:₩Python₩Python312'로 변경한 뒤 [install]을 눌러 설치한다.

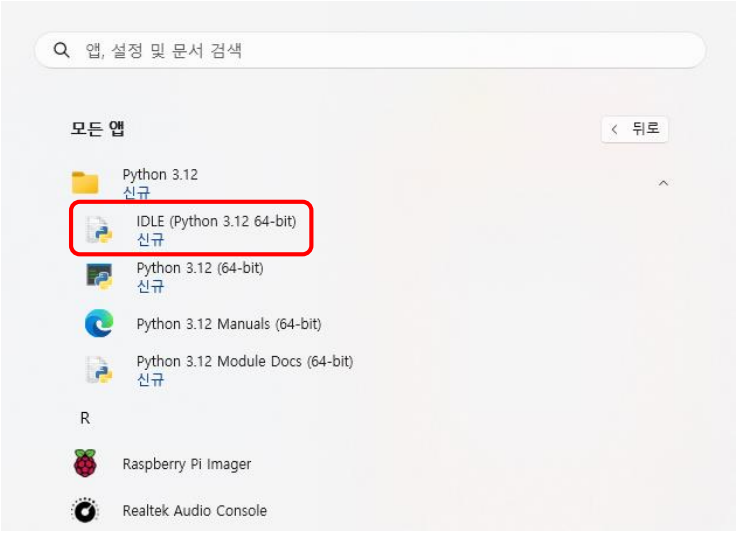




파이썬 사용 방법

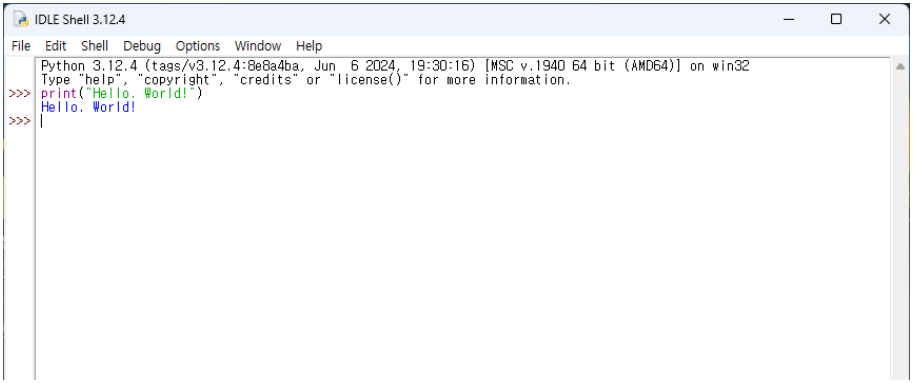
파이썬을 설치하고 난 뒤에는 파이썬 프로그램을 코딩할 줄 알아야 한다.
파이썬에는 프로그램을 개발할 수 있는 환경이 포함되어 있다. 이를 IDLE(Integrated DeveLopment Environment, 통합 개발 환경)이라고 한다.

이번에는 IDLE을 사용해서 간단한 파이썬 프로그램을 만들어 보도록 하자.
windows의 시작 버튼을 클릭하고 [모든 앱]에서 [Python 3.12]아래에 [IDLE (Python 3.12 64-bit)]를 클릭한다.

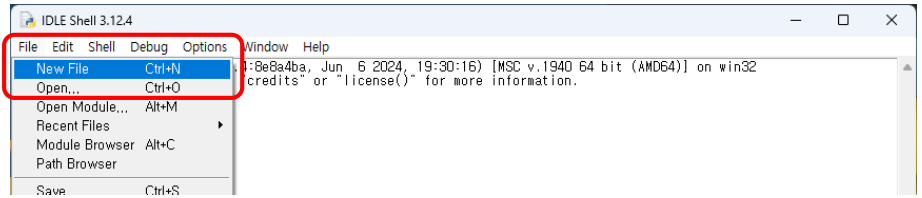


파이썬 사용 방법

IDLE 창이 실행되면 프롬프트(Prompt)라고 하는 >>>을 볼 수 있는데, 이곳이 명령 코드를 입력하는 입력 줄이다. 여기에 print("Hello. World!")를 입력하고 enter를 누르면 실행 결과로 Hello. World! 가 출력되는 것을 확인할 수 있다.
이렇게 명령을 한 줄 입력하면 바로 실행 결과를 보여주는 작업 모드는 "Shell"이라는 대화형 모드이다.

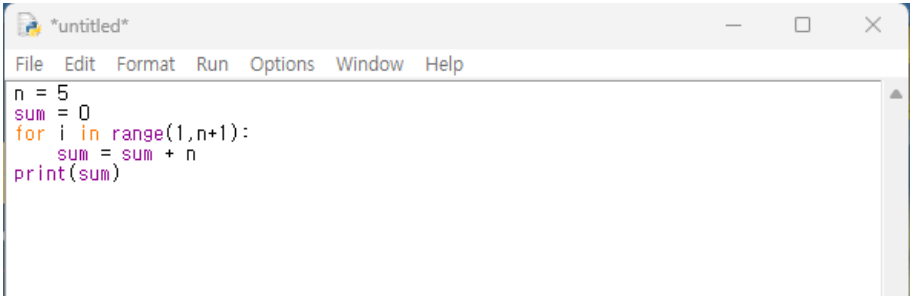


하지만, 코드를 여러 줄 작성하고 한번에 실행하고 싶을 수 있는데, 이럴 때는 스크립트 모드를 사용하면 된다. IDLE 창의 상단 메뉴의 [File]에서 [New File]을 클릭하면 스크립트 모드의 창이 나타난다.

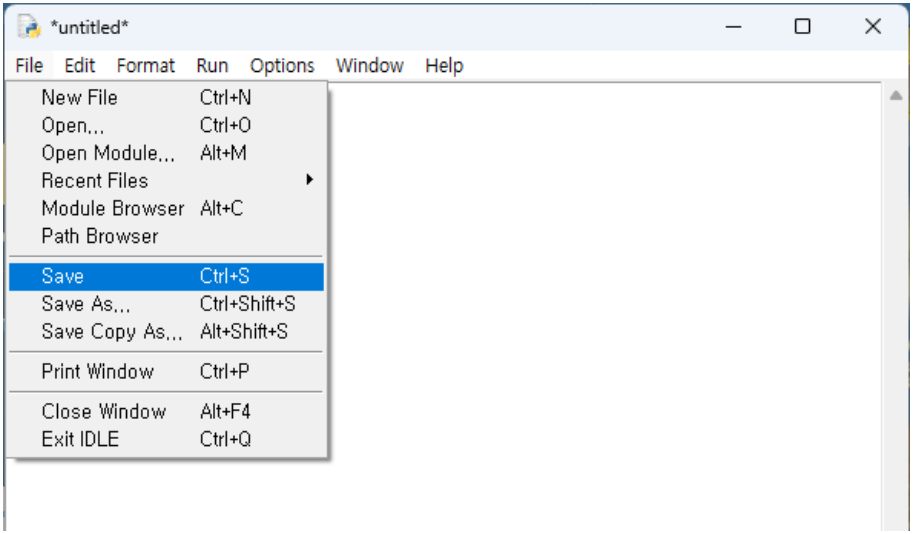


파이썬 사용 방법

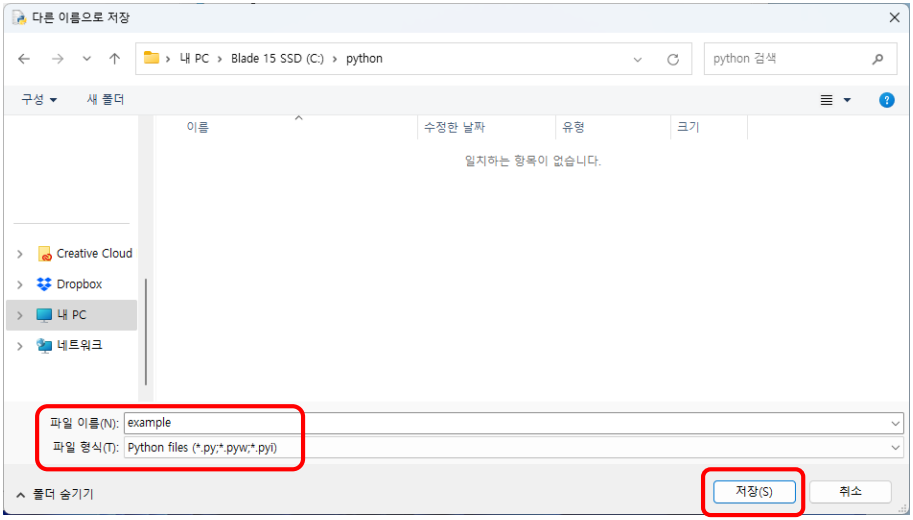
여러 줄의 코드를 작성해보기 위해서 앞에서 알고리즘 복잡도 분석 파트에서 사용했던 코드를 활용해 보자.

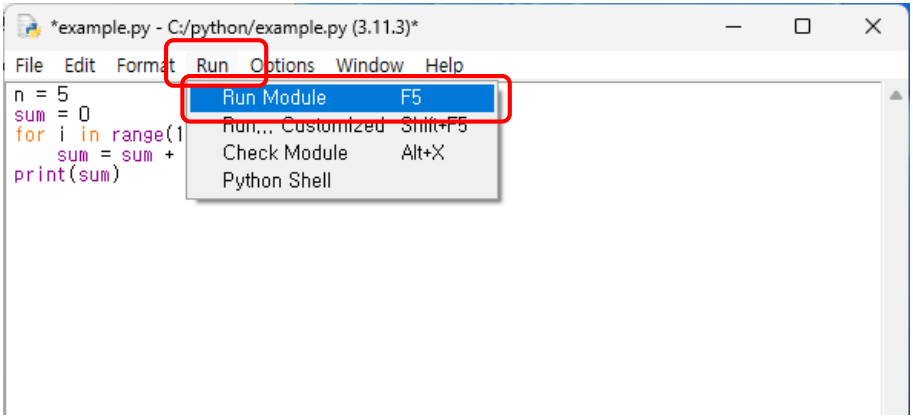


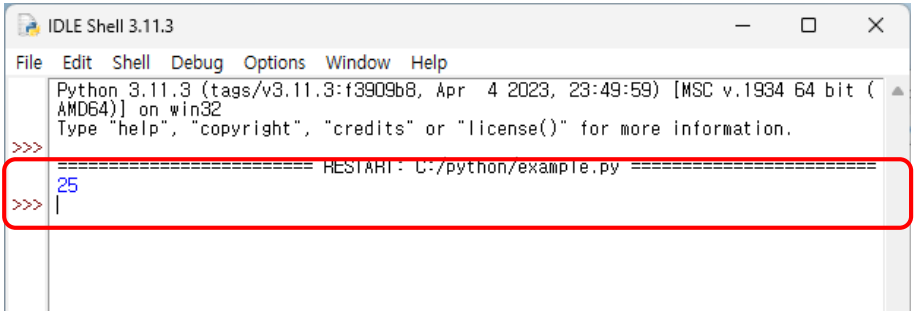
위에 보이는 것처럼 코드를 작성한 뒤에 [File]에서 [Save]를 클릭하고 파일을 저장한다. 이때, 파일의 확장자명은 “.py”가 된다.



파이썬 사용 방법


다른 이름으로 저장
내 PC › Blade 15 SSD (C:) › python
python 검색
구성
새 폴더
이름
수정한 날짜
유형
크기
일치하는 항목이 없습니다.
Creative Cloud
Dropbox
내 PC
네트워크
파일 이름(N): example
파일 형식(T): Python files (*.py;*.pyw;*.pyi)
폴더 숨기기
저장(S)
취소


example.py - C:/python/example.py (3.11.3)
File Edit Format Run Options Window Help
n = 5
sum = 0
for i in range(1
sum = sum +
print(sum)
Run Module F5
Run... Customized Shift+F5
Check Module Alt+X
Python Shell


IDLE Shell 3.11.3
File Edit Shell Debug Options Window Help
Python 3.11.3 (tags/v3.11.3:f3909b8, Apr 4 2023, 23:49:59) [MSC v.1934 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
========== RESTART: C:/python/example.py ==========
25
>>>

제 2 장

파이썬 기초

1) 기본 문법과 데이터형
2) 연산자
3) 제어문
4) 함수
5) 예외 처리

01. 기본 문법과 자료형

주석(Comment)이란?

2-1. 주석

```
1  # 파이썬은 인간다운 언어이다
2  # 문법이 쉬워 빠르게 배울 수 있다
3  # 무료이지만 강력하다
4  # 간결하다
5  # 프로그래밍을 즐기게 해 준다
6  # 개발 속도가 빠르다
7  # Life is too short, You need python.
```

- 파이썬 주석처리: #

주석은 실제 코드가 아니므로 주석 처리 된 부분은 실행되지 않는다.

변수?

2-2. 변수 선언

```
1  int1 = 7
2  float1 = 3.14
3  bool1 = True
4  str1 = "Hi"
5
6  int2 , float2 , bool2 , str2 = 10, 2.28, False, "Hello"
```

✓ **1~4행** 한 줄마다 변수 선언을 한다.

✓ **6행** 한 줄에 모든 변수 선언이 가능하다.

입력과 출력 함수?

2-3. print() 함수

```
1  print("Hello World!")
2  print("100")
3  print("%d" % 50)
4  print(30)
5  print("Hello/nWorld")
```

출력 결과

```
Hello World!
100
50
30
Hello
World
```

- ✓ **1행** 문자열 Hello World를 출력한다.
- ✓ **2행** 문자열 100을 출력한다.
- ✓ **3행** 숫자 50을 출력한다.
- ✓ **4행** 숫자 30을 출력한다.
- ✓ **5행** new line을 의미하는 /n로 해당 문자열을 줄바꿈하여 출력한다.

출력 함수인 print() 함수를 통해 변수 등 여러 값을 출력할 수 있다.

위 예시 외에도 끝 문자 설정, 구분자 변경, 문자열 포맷팅 등 다양한 옵션이 있다.

- print() 함수에서 사용 가능한 서식

서식	값의 예	설명
%d, %x, %o	10, 100, 1234	정수(10진수, 16진수, 8진수)
%f	0.5, 1.0, 3.14, 3.14e3	실수(소수점이 붙은 수)
%c	'b', '한', '파'	문자 하나로 된 글자
%s	"안녕", 'abcdefg', "a"	한 글자 이상의 문자열

- 특수 문자 처리 방법

특수문자	설명	예시 코드	출력 결과
/n	줄 바꿈	print("Hello/nWorld")	Hello World
/t	탭	print("Hello/tWorld")	Hello World
//	백슬래시 출력	print("This is a backslash: //")	This is a backslash: /
/'	작은따옴표 출력	print('It/'s a book')	It's a book
/"	큰따옴표 출력	print("He said /"Hello/"")	He said "Hello"
/r	캐리지 리턴	print("Hello/rWorld")	World
/b	백스페이스	print("Hello/bWorld")	HellWorld

02. 기본 문법과 자료형

2-4. input() 함수

```
1  name = input("Enter your name: ")
2  print("Hello, " + name + "!")
3
4  age = int(input("Enter your age: "))
5  print(f"Next year, you will be {age + 1} years old.")
```

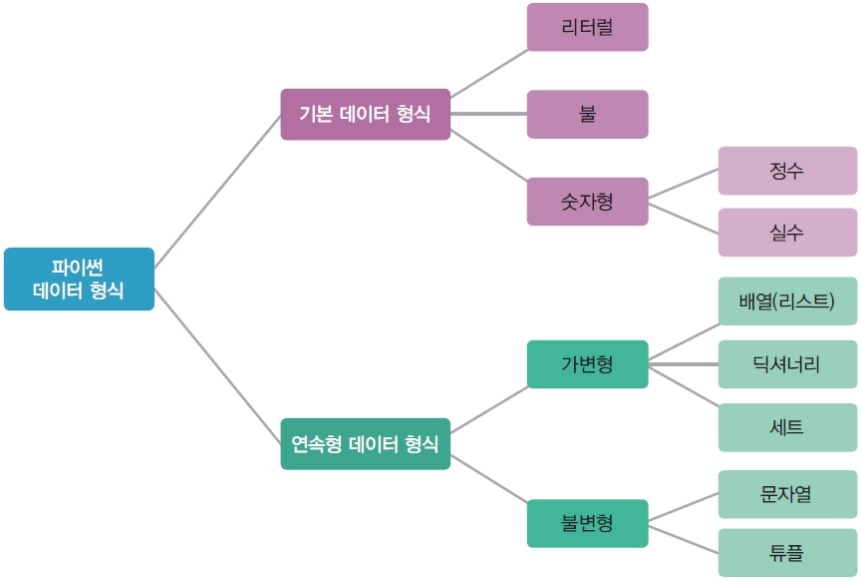
출력 결과

```
Enter your name: Hong Gil Dong
Hello, Hong Gil Dong!
Enter your age: 20
Next year, you will be 21 years old.
```

- ✓ **1행** name 변수에 사용자의 입력을 문자열 형태로 저장한다.
- ✓ **2행** 입력된 문자열을 출력한다.
- ✓ **4행** age 변수에 사용자의 입력을 정수형으로 저장한다.
- ✓ **5행** 문자열 포맷팅을 사용해 다음 해 나이를 출력한다.

입력 함수인 input() 함수는 사용자로부터 입력을 받기 위해 사용된다. 항상 문자열을 반환하기 때문에, 필요에 따라 형 변환을 수행해야 한다.

자료형이란?



기본 데이터형

1) **리터럴(Literal)**: 숫자, 문자열 등 상수 값을 직접 표기한 데이터다.

2) **불(bool)**: 참/거짓으로 저장되는 데이터, 조건문의 조건식 결과로 사용된다.

bool

```
1  boolVar = 100 > 200
2  boolVar
3  type(boolVar)
```

출력 결과

```
bool
```

3) 숫자형

- 정수형(int): 소수점이 없는 숫자를 나타낸다.
- 실수형(float): 소수점을 포함한 숫자를 나타낸다.

정수형

```
1 intVar = 100 * 3
2 print(intVar)
3 type(intVar)
```

출력 결과

```
300
int
```

실수형

```
1 floatVar = 2.35212
2 print(floatVar)
3 type(floatVar)
```

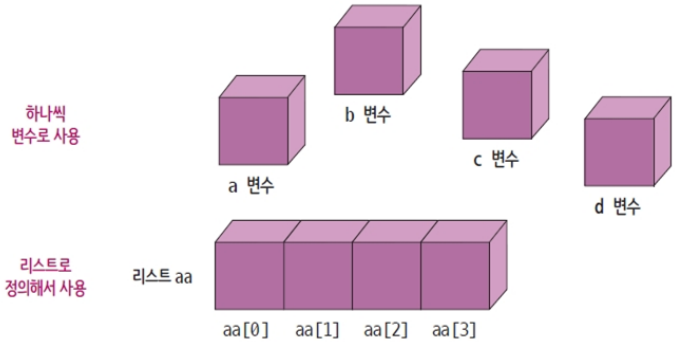
출력 결과

```
2.35212
float
```

연속 데이터형

1) 가변형

- 배열(리스트)
    - 변수를 한 줄로 붙인 후 전체에 이름(aa)을 지정한다.
    - aa[0], aa[1], aa[2], aa[3]처럼 번호를 붙여 사용한다.



리스트의 개념

1차원 리스트

```
1   aa = []
2   for i in range(0, 4):
3       aa.append(0)
4   total = 0
5
6   for i in range(0, 4):
7       aa[i] = int(input(str(i + 1) + "번째 숫자 : "))
8
9   for i in range(0, 4):
10      total = total + aa[i]
11
12  print("합계 ==> %d" % total)
```

출력 결과

```
1번째 숫자 : 20
2번째 숫자 : 10
3번째 숫자 : 30
4번째 숫자 : 40 합계 ==> 100
```

- ✓ **1~3행** 빈 리스트 aa를 생성하고, 네 번의 반복으로 aa에 0을 추가한다.
- ✓ 4행 합계를 저장할 변수 total을 0으로 초기화한다.
- ✓ **6~7행** 네 번의 반복 동안 사용자로부터 입력받은 숫자를 정수 형태로 리스트 aa[i]에 저장한다.
- ✓ **9~10행** 네 번의 반복 동안 리스트 aa[i]의 각 인덱스의 값을 total에 저장한다.
- ✓ 12행 최종 합계를 출력한다.

- **리스트 조작 함수**

함수	설명	사용법
append()	리스트 맨 뒤에 항목을 추가한다.	리스트명.append(값)
pop()	리스트 맨 뒤의 항목을 빼낸다(리스트에서 해당 항목을 삭제한다).	리스트명.pop()
sort()	리스트의 항목을 정렬한다.	리스트명.sort()
reverse()	리스트 항목의 순서를 역순으로 만든다.	리스트명.reverse()
index()	지정한 값을 찾아 위치를 반환한다.	리스트명.index(찾을값)
insert()	지정한 위치에 값을 삽입한다.	리스트명.insert(위치, 값)

remove()	리스트에서 지정한 값을 제거한다. 단 지정한 값이 여러 개이면 첫 번째 값만 지운다.	리스트명.remove(지울값)
extend()	리스트 뒤에 리스트를 추가한다. 리스트의 더하기(+) 연산과 기능이 동일하다.	리스트명.extend(추가할리스트)
count()	리스트에서 해당 값의 개수를 센다.	리스트명.count(찾을값)
len()	리스트에 포함된 전체 함목의 개수를 센다.	len(리스트명)
sorted()	리스트의 함목을 정렬해서 새로운 리스트에 대입한다.	새리스트 = sorted(리스트)

2차원 리스트

```
1  list1 = []
2  list2 = []
3  value = 1
4
5  for i in range(0, 3):
6      for k in range(0, 4):
7          list1.append(value)
8          value += 1
9      list2.append(list1)
10     list1 = []
11
12 for i in range(0, 3):
13     for k in range(0, 4):
14         print("%3d" % list2[i][k], end=" ")
15     print(" ")
```

출력 결과

```
1 2 3 4
5 6 7 8
9 10 11 12
```

✓ 1~3행 list1, list2라는 빈 리스트를 생성하고 value를 1로 초기화한다.

✓ **5~10행** 3번의 반복문은 행, 4번의 반복문은 열을 나타낸다. value 값을 list1에 추가한 뒤, value를 1 증가시켜서 각 행에 4개의 값을 추가한다.

✓ **12~15행** 생성된 2차원 리스트를 출력한다.

- 리스트 컴프리헨션(함축)

파이썬에서 리스트를 간결하고 효율적으로 생성할 수 있는 문법이다. 반복문을 사용하여 리스트를 생성하는 것보다 더 짧은 코드로 작성이 가능하다.

기본 문법은 **[표현식 for 함목 in range() if 조건식]** 이다.

리스트 컴프리헨션 예시

```
1 numList = [num * num for num in range(1, 6)]
2 print(numList)
```

출력 결과

```
[1, 4, 9, 16, 25]
```

✓ **1행** 리스트 컴프리헨션으로 1~5까지의 제곱을 포함하는 리스트를 생성한다.

✓ **2행** 실행 결과를 출력한다.

- 딕셔너리

딕셔너리(Dictionary)는 키와 값을 중괄호 { }로 묶어 구성하는 자료구조를 말한다.

기본 문법은 **딕셔너리변수 = {키1:값1, 키2:값2, 키3:값3, ...}** 이다.

딕셔너리에서 키는 중복되면 안 된다는 특성이 있다.

키	값
학번	1004
이름	홍길동
학과	컴퓨터공학과

홍길동 학생의 정보

위의 표를 딕셔너리로 표현하면 아래와 같다.

딕셔너리 예시 1

```
1  student1 = {'학번': 1004, '이름': '홍길동', '학과': '컴퓨터공학과'}
2  print(student1)
```

출력 결과

```
{'학번': 1004, '이름': '홍길동', '학과': '컴퓨터공학과'}
```

✓ **1행** student1이라는 딕셔너리에 세 개의 키-값 쌍을 생성한다.

✓ **2행** 실행 결과를 출력한다.

딕셔너리 예시 2

```
1  student1 = {'학번': 1000, '이름': '홍길동', '학과': '컴퓨터학과'}
2  print(student1)
3
4  student1['연락처'] = '010-1111-2222'
5  print(student1)
6
7  student1['학과'] = '파이썬학과'
8  print(student1)
9
10 del(student1['학과'])
11 print(student1)
12
13 student1 = {'학번': 1000, '이름': '홍길동', '학과': '파이썬학과', '학번': 2000}
14 print(student1)
15
16 print(student1['학번'])
17 print(student1.get('이름'))
18
19 items = student1.items()
20 print(items)
```

출력 결과

```
{'학번': 1000, '이름': '홍길동', '학과': '컴퓨터학과'}
{'학번': 1000, '이름': '홍길동', '학과': '컴퓨터학과', '연락처': '010-1111-2222'}
{'학번': 1000, '이름': '홍길동', '학과': '파이썬학과', '연락처': '010-1111-2222'}
{'학번': 1000, '이름': '홍길동', '연락처': '010-1111-2222'}
{'학번': 2000, '이름': '홍길동', '학과': '파이썬학과'}
2000
홍길동
dict_items([('학번', 2000), ('이름', '홍길동'), ('학과', '파이썬학과')])
```

- ✓ **1~2행** 키-값 쌍이 3개인 딕셔너리를 생성하여 출력한다.
- ✓ **4~5행** 딕셔너리에 새로운 키-값 쌍을 추가하여 출력한다.
- ✓ **7~8행** '학과 키의 값을 파이썬학과로 수정하고 딕셔너리를 출력한다.

- ✓ **10~11행** 학과의 키-값 쌍을 삭제하고 딕셔너리를 출력한다.
- ✓ **13~14행** 딕셔너리를 다시 생성한다. 중복된 학번 키의 경우 마지막 값인 2000으로 덮어씌워진다.
- ✓ **16~17행** 딕셔너리의 값에 접근하여 딕셔너리 값을 출력한다.
- ✓ **19~20행** 딕셔너리의 모든 키-값 쌍을 반환하여 출력한다.

- 세트

세트(set)는 키만 모아 놓은 딕셔너리의 특수한 형태이다. 딕셔너리의 키는 중복되면 안 된다는 특성이 있기 대문에 세트에 들어 있는 값은 항상 유일해야만 한다. 세트 생성을 위해서는 중괄호 { }를 사용하고, : 없이 값을 입력한다.

여러 개의 세트가 있는 경우 교집합, 합집합, 차집합, 대칭 차집합을 구할 수 있다.

세트 예시

```
1  mySet1 = {1, 2, 3, 3, 3, 4}
2  print(mySet1)
3
4  mySet1 = {1, 2, 3, 4, 5}
5  mySet2 = {4, 5, 6, 7}
6
7  print(mySet1 & mySet2)
8  print(mySet1.intersection(mySet2))
9
10 print(mySet1 | mySet2)
11 print(mySet1.union(mySet2))
12
13 print(mySet1 - mySet2)
14 print(mySet1.difference(mySet2))
15
16 print(mySet1 ^ mySet2)
17 print(mySet1.symmetric_difference(mySet2))
```

출력 결과

```
{1, 2, 3, 4}
{4, 5}
{4, 5}
{1, 2, 3, 4, 5, 6, 7}
{1, 2, 3, 4, 5, 6, 7}
{1, 2, 3}
{1, 2, 3}
{1, 2, 3, 6, 7}
{1, 2, 3, 6, 7}
```

- ✓ **1~2행** 중복된 값이 제거된 후 출력된다.
- ✓ **4~5행** mySet1, mySet2라는 두 세트를 정의한다.
- ✓ **7~8행** 두 세트 간 교집합을 출력한다.
- ✓ **10~11행** 두 세트 간 합집합을 출력한다.
- ✓ **13~14행** 두 세트 간 차집합을 출력한다.
- ✓ **16~17행** 두 세트 간 대칭 차집합을 출력한다.

2) 불변형

- 문자열

문자열은 문자들의 연속으로, 파이썬에서는 큰따옴표(“ “)나 작은따옴표(‘ ‘)로 묶어서 표현한다. 문자열은 불변형 데이터 형식으로, 한 번 생성된 문자열은 변경이 불가능하다.

문자열 예시

```
1  ss = "자료구조와 알고리즘"
2  print(ss[0])
3  print(ss[1:4])
4
5  ss = "파이썬" + " 최고!"
6  print(ss)
7
8  ss = "파이썬" * 3
9  print(ss)
10
11 ss = '파이썬 abc'
12 print(len(ss))
13
14 ss = "파이썬 공부는 즐겁습니다. 모두들 파이썬을 사랑해요."
15 print(ss.count('파'))
16 print(ss.find('공부'))
17 print(ss.index('파이썬'))
18
19 ss = "하나:둘:셋"
20 print(ss.split(':'))
21
22 ss = '!'.join(['Python', '열심히', '공부', '중'])
23 print(ss)
24
25 before = ['2022', '12', '31']
26 after = list(map(int, before))
27 print(after)
```

출력 결과

```
자
료구조
파이썬 최고!
파이썬파이썬파이썬
7
2
4
```

```
0
['하나', '둘', '셋']
Python!열심히!공부!중
[2022, 12, 31]
```

- ✓ **1~2행** 문자열 ss를 자료구조와 알고리즘으로 초기화하여 문자열의 0번째 인덱스, 1번째~3번째 인덱스 값을 출력한다.
- ✓ **5~6행** 두 문자열을 연결하여 변수 ss에 할당하고 출력한다.
- ✓ **8~9행** 파이썬 문자열을 세 번 반복하여 변수 ss에 할당하고 출력한다.
- ✓ **11~12행** 문자열 변수 ss를 새로 할당하여 문자열의 길이를 출력한다.
- ✓ **15행** 문자열 ss에서 '파'가 나온 개수를 세어 출력한다.
- ✓ **16행** 문자열 ss에서 '공부' 문자열이 처음으로 등장하는 위치를 출력한다.
- ✓ **17행** 문자열 ss에서 '파이썬' 문자열이 처음으로 등장하는 위치를 출력한다.
- ✓ **19~20행** 문자열 ss를 콜론(:)으로 분리하여 리스트로 변환 후 출력한다.
- ✓ **22~23행** 리스트의요소들을 느낌표(!) 구분자로 결합하여 하나의 문자열로 만들어 출력한다.
- ✓ **25~27행** 문자열로 구성된 리스트 befor의 각 요소를 정수형으로 변환하여 새로운 리스트 after에 저장하여 출력한다.

- 튜플

튜플은 파이썬에서 리스트와 비슷하지만 약간 다른 자료형이다. 튜플은 값을 변경할 수 없는 불변형 데이터 형식으로, 한 번 생성되면 값을 수정할 수 없다. 주로 읽기 전용 데이터 또는 변경할 필요가 없는 데이터를 저장할 때 사용한다.

문자열 예시

```
1  tuple1 = (10, 20, 30)
2  print(tuple1)
3
4  tuple1 = (10, 20, 30, 40)
5  print(tuple1[0])  # 10
6  print(tuple1[1] + tuple1[2])  # 50
7
8  print(tuple1[1:3])  # (20, 30)
9  print(tuple1[:2])  # (10, 20)
10 print(tuple1[:])  # (10, 20, 30, 40)
11
12 tuple2 = ('A', 'B')
13 print(tuple1 + tuple2)  # (10, 20, 30, 40, 'A', 'B')
14 print(tuple2 * 3)  # ('A', 'B', 'A', 'B', 'A', 'B')
15
16 myTuple = (10, 20, 30)
17 myList = list(myTuple)
18 myList.append(40)
19 myTuple = tuple(myList)
20 print(myTuple)  # (10, 20, 30, 40)
```

출력 결과

```
(10, 20, 30)
10
50
(20, 30)
(10, 20)
(10, 20, 30, 40)
(10, 20, 30, 40, 'A', 'B')
('A', 'B', 'A', 'B', 'A', 'B')
(10, 20, 30, 40)
```

- ✓ **1~2행** tuple1이라는 이름의 튜플을 생성하고 출력한다.
- ✓ **4~5행** tuple1의 값을 재정의하고 각 튜플의 인덱스의 값을 출력한다.
- ✓ **6행** tuple1의 1번 인덱스와 2번 인덱스의 값을 더한 값을 출력한다.
- ✓ **8~10행** 각 튜플의 인덱스를 슬라이싱하여 요소를 출력한다.
- ✓ **12~13행** tuple2라는 이름의 튜플을 생성하고 tuple1과 tuple2의 요소를 더한 값을 출력한다.
- ✓ **14행** tuple2를 세 번 반복한 값을 출력한다.
- ✓ **16~20행** myTuple이라는 이름의 튜플을 생성한 뒤 리스트로 변환하여 myList에 저장한다. myList의 마지막 인덱스에 40을 추가한 뒤, 이를 다시 튜플로 변환하여 mytuplr에 저장한 값을 출력한다.

02. 연산자

산술 연산자란?

연산자	설명	사용 예	
=	대입 연산자	a=3	정수 3을 a에 대입
+	더하기	a=5+3	5와 3을 더한 값을 a에 대입
-	빼기	a=5-3	5와 3을 뺀 값을 a에 대입
*	곱하기	a=5*3	5와 3을 곱한 값을 a에 대입
/	나누기	a=5/3	5를 3으로 나눈 값을 a에 대입
//	나누기(몫)	a=5//3	5를 3으로 나눈 후 소수점을 버린 값을 a에 대입
%	나머지 값	a=5%3	5를 3으로 나눈 후 나머지 값을 a에 대입
**	제곱	a=5**3	5의 3제곱을 a에 대입

3-1. 산술 연산자 사용 예

```
1 a = 4
2 b = 3
3 print(a+b, a-b, a*b, a/b, a//b, a%b, a**b)
```

출력 결과

```
7 1 12 1.3333333333333333 1 1 64
```

대입 연산자란?

연산자	사용 예	
+=	a+=3	a=a+3과 동일
-=	a-=3	a=a-3과 동일
*=	a*=3	a=a*3과 동일
/=	a/=3	a=a/3과 동일
//=	a//=3	a=a//3과 동일
%=	a%=3	a=a%3과 동일
**=	a**=3	a=a**3과 동일

파이썬의 대입 연산자는 변수에 값을 할당하는 데 사용되며, 종종 다른 연산과 결합되어 사용된다. 이 연산자들은 변수의 현재 값을 기반으로 연산을 수행한 후, 그 결과를 동일한 변수에 저장한다.

대입 연산자는 코드를 더 간결하고 읽기 쉽게 만들어준다. 이러한 연산자들을 이해하고 활용하면, 다양한 상황에서 변수 값을 효율적으로 업데이트할 수 있다.

3-2. 대입 연산자 사용 예

```
1  a = 5
2  a += 3
3  print(a)  # 출력: 8
4
5  a = 5
6  a *= 3
7  print(a)  # 출력: 15
8
9  a = 5
10 a /= 3
11 print(a)  # 출력: 1.6666666666666667
12
13 a = 5
14 a //= 3
15 print(a)  # 출력: 1
16
17 a = 5
18 a %= 3
19 print(a)  # 출력: 2
20
21 a = 5
22 a **= 3
23 print(a)  # 출력: 125
```

출력 결과

```
8
15
1.6666666666666667
1
2
125
```

관계|비교 연산자란?

개념	연산자	의미	설명
a ㅁ b = [참 : True / 거짓: False	==	같다.	두 값이 동일하면 참
	!=	같지 않다.	두 값이 다르면 참
	>	크다.	왼쪽이 크면 참
	<	작다.	왼쪽이 작으면 참
	>=	크거나 작다.	왼쪽이 크거나 같으면 참
	<=	작거나 같다.	왼쪽이 작거나 같으면 참

3-3. 관계|비교 연산자 예시

```
1  a, b = 10, 20
2  print(a == b, a != b, a > b, a < b, a >= b, a <= b)
```

출력 결과

```
False True False True False True
```

논리 연산자란?

연산자	예시	설명
and	a and b	a와 b가 모두 참이면 참
or	a or b	a 또는 b 중 하나라도 참이면 참
not	not a	a가 참이면 거짓, 거짓이면 참

비트 논리 연산자란?

연산자	예시	설명
&	a & b	A와 b의 비트 AND
|	a | b	A와 b의 비트 OR
^	a ^ b	A와 B의 비트 XOR
~	~a	a의 비트 NOT
<<	a << 1	a의 비트를 왼쪽으로 1비트 시프트
>>	a >> 1	a의 비트를 오른쪽으로 1비트 시프트

비트 논리 연산자의 예시

- 비트 AND 연산

```
  1101
& 1011
------
  1001
```

비트 단위로 AND 연산을 한다. 두 개의 비트 모두 1이면 1을 반환한다.

- 비트 OR 연산

```
  1101
| 1011
------
  1111
```

비트 단위로 OR 연산을 한다. 두 개의 비트 중 하나라도 1이면 1을 반환한다.

비트 논리 연산자의 예시

- 비트 XOR 연산

```
  1101
^ 1011
------
  0110
```

비트 단위로 XOR 연산을 한다. 두 개의 비트가 서로 다르면 1을 반환한다.

- 비트 NOT 연산

```
~ 1101
------
  0010
```

비트 단위로 NOT 연산을 한다. 0인 비트는 1로, 1인 비트는 0으로 반환한다.

비트 논리 연산자의 예시

- **비트 Left Shift 연산**

1101 << 1 → 11010

비트 단위로 Left Shift 연산을 한다. 위의 예시는 왼쪽으로 1bit만큼 이동한다. 이진수인 1101은 십진수로 13이므로 13 << 1 = 26이 된다.

- **비트 Right Shift 연산**

1101 >> 1 → 0110

비트 단위로 Right Shift 연산을 한다. 위의 예시는 오른쪽으로 1bit만큼 이동한다. 이진수인 1101은 십진수로 13이므로 13 >> 1 = 6이 된다.

3-4. 논리 연산자 예시

```
1  a = True
2  b = False
3
4  print(a and b)
5  print(a or b)
6  print(not a)
```

출력 결과

```
False
True
False
```

- ✓ **4행** a와 b가 모두 True가 아니기 때문에 False를 출력한다.
- ✓ **5행** a와 b 둘 중 a가 True이기 때문에 True를 출력한다.
- ✓ **6행** a가 True이기 때문에 반대 값인 False를 출력한다.

3-5. 논리 연산자 예시

```
1  a = 0b1101
2  b = 0b1010
3
4  print(bin(a & b))
5  print(bin(a | b))
6  print(bin(a ^ b))
7  print(bin(~a))
8
9  print(bin(a << 1))
10 print(bin(a >> 1))
```

출력 결과

```
0b1001
0b1111
0b110
-0b1110
0b11010
0b110
```

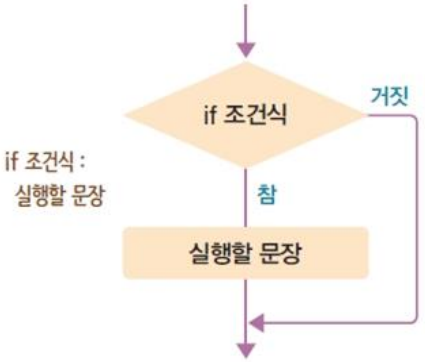
- ✓ **1~2행** a=13, b=10의 이진수 값을 대입한다.
- ✓ **4행** a와 b의 비트 논리 AND 연산 값을 출력한다.
- ✓ **5행** a와 b의 비트 논리 OR 연산 값을 출력한다.
- ✓ **6행** a와 b의 비트 논리 XOR 연산 값을 출력한다.
- ✓ **7행** a의 NOT 연산(2의 보수) 값을 출력한다.
- ✓ **9행** a를 왼쪽으로 1비트 시프트 수행한 값을 출력한다.
- ✓ **10행** a를 오른쪽으로 1비트 시프트 수행한 값을 출력한다.

03. 제어문

제어문은 프로그램의 흐름을 효율적으로 이용하기 위해 사용한다.

조건문이란?

조건문이란 해당 조건이 참인 경우 실행되는 문장을 의미한다.



if문의 형식과 순서도

만약 조건식의 내용이 참이라면 문장의 내용이 실행되고, 거짓이라면 문장의 내용이 실행되지 않는다. 다음과 같은 예시를 고려해보자. 밖에 비가 온다면 우산을 챙기고, 그렇지 않다면 우산을 챙기지 않는다. 이것을 조건문으로 표현해 보면 어떻게 될지 생각해보자.

4-1. if-else문

```
1  rain = True
2  if rain:
3      print("우산을 챙겨라")
4  else:
5      print("우산을 챙기지 말아라")
```

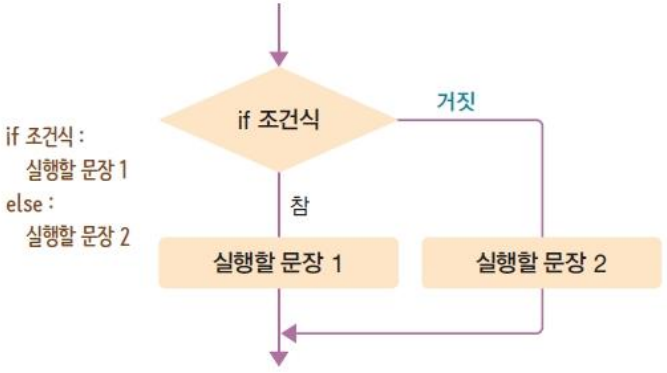
출력 결과

우산을 챙겨라

✓ **1행** 비가 오는 상황이라고 가정하여 rain=True로 설정한다.

✓ **2~3행** 만약 비가 온다면, “우산을 챙겨라”가 출력된다. rain=True이기 때문에 해당 조건문이 출력된다.

✓ **4~5행** 비가 오지 않는다면, “우산을 챙기지 말아라”가 출력된다. rain=True이기 때문에 이 줄의 코드는 실행되지 않는다.



if-else 문의 형식과 순서도

그러나, 세상에는 두 가지의 조건만으로는 표현이 되지 않는 경우가 많다. 예를 들어 성적을 분류할 때 90점 이상이면 A, 80점 이상이면 B, 70점 이상이면 C, 60점 이상이면 D, 그 외 학점은 F로 분류해야 해야 하는데, 'elif'를 사용해서 해당 기능을 구현할 수 있다.

4-2. if-elif-else문

```
1   score = 85  # 예시 점수
2
3   if score >= 90:
4       grade = 'A'
5   elif score >= 80:
6       grade = 'B'
7   elif score >= 70:
8       grade = 'C'
9   elif score >= 60:
10      grade = 'D'
11  else:
12      grade = 'F'
13
14  print(f"점수: {score}, 학점: {grade}")
```

출력 결과

```
점수: 85, 학점: B
```

- ✓ **1행** score 변수에 점수를 할당한다.
- ✓ **3행** 첫번째 if문에서 score가 90점 이상인지를 검사한다. 참인 경우 'A'를 할당하고, 조건 검사를 종료한다.
- ✓ **5~10행** 첫 번째 조건이 거짓이면 elif문이 순차적으로 실행된다. score가 80점 이상인 경우 B, 70점 이상인 경우 C, 60점 이상인 경우 D를 할당한다.

✓ **11행** 모든 elif 조건이 거짓인 경우, else 블록이 실행되어 F를 할당한다.

✓ **14행** score은 85점으로 5행의 조건문에 해당하여 B학점을 출력한다.

이처럼 'elif'를 사용하면 여러 조건을 체계적으로 처리할 수 있어 복잡한 조건 분기를 쉽게 구현할 수 있다.

반복문

반복문은 특정 코드를 여러 번 실행해야 할 때 유용하다. 파이썬에서는 두 가지 반복문인 'for'문과 'while'문을 제공한다. 각각의 반복문은 다른 상황에서 더 적합하게 사용된다.

for문

주로 리스트, 튜플, 문자열 등 시퀀스를 순회할 때 사용된다. for문의 기본 구조는 아래와 같다. 리스트나 튜플, 문자열의 첫 번째 요소부터 마지막 요소까지 차례대로 변수에 대입되어 '수행할_문장1', '수행할_문장2' 등이 수행된다.

```
for 변수 in 리스트(또는 튜플, 문자열):
    수행할_문장1
    수행할_문장2
    ...
```

반복 횟수가 명확할 때 사용하기 좋다. 주로 리스트, 튜플, 문자열 등 시퀀스를 순회할 때 사용된다. for문의 기본 구조는 아래와 같다. 리스트나 튜플, 문자열의 첫 번째 요소부터 마지막 요소까지 차례대로 변수에 대입되어 '수행할_문장1', '수행할_문장2' 등이 수행된다.

```
>>> test_list = ['one', 'two', 'three']
>>> for i in test_list:
...     print(i)
...
one
two
three
```

전형적인 for문

['one', 'two', 'three'] 리스트의 첫 번째 요소인 'one'이 먼저 i 변수에 대입된 후 print(i) 문장을 수행한다. 다음에 두 번째 요소 two가 i 변수에 대입된 후 print(i) 문장을 수행하고 리스트의 마지막 요소까지 이를 반복한다.

```
>>> a = [(1,2), (3,4), (5,6)]
>>> for (first, last) in a:
...     print(first + last)
...
3
7
11
```

튜플 리스트인 경우

튜플 리스트인 경우, a 리스트의 요소가 튜플이기 때문에 각각의 요소가 자동으로 (first, last) 변수에 대입되어 실행된다.

for문의 개념에 대해 알아보았으니, 이를 응용한 예제들을 살펴보도록 하자.

4-3. for문으로 구구단 만들기

```
1  for i in range(1, 10):
2      for j in range(1, 10):
3          print(i, "x", j, "=", i * j)
4      print("-" * 15)
```

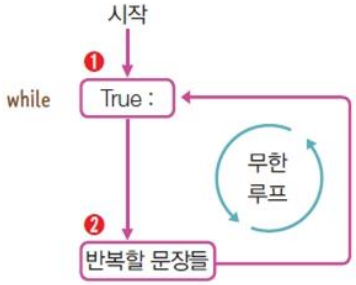
출력 결과

```
1 x 1 = 1
1 x 2 = 2
1 x 3 = 3
1 x 4 = 4
1 x 5 = 5
1 x 6 = 6
1 x 7 = 7
1 x 8 = 8
1 x 9 = 9
---------------
2 x 1 = 2
2 x 2 = 4
  (중략)
9 x 5 = 45
9 x 6 = 54
9 x 7 = 63
9 x 8 = 72
9 x 9 = 81
---------------
```

- ✓ **1행** 1부터 9까지의 숫자를 생성해 순차적으로 변수 i에 할당하며 반복한다.
- ✓ **2행** 1부터 9까지의 숫자를 생성해 순차적으로 변수 j에 할당하며 반복한다.
- ✓ **3행** i가 1인 경우일 때, j를 1부터 9까지 곱해준다(1의 곱셈), i가 2인 경우에도 j를 1부터 9까지 곱해준다. 이것을 i가 9일 때까지 반복한다.
- ✓ **4행** 각 구구단 단을 구분하기 위해 임의의 하이픈 문자열을 출력한다.

while문

조건문이 참인 동안 while문에 속한 코드가 반복해서 실행된다.



while 문의 형식과 무한 루프

```
while 조건문:
    수행할_문장1
    수행할_문장2
    수행할_문장3
    ...
```

4-4. while문으로 1부터 9까지 더하기

```
1  total = 0
2  i = 1
3
4  while i <= 9:
5      total += i
6      i += 1
7
8  print("1부터 9까지 더한 값:", total)
```

출력 결과

```
1부터 9까지 더한 값: 45
```

- ✓ **1~2행** 총합을 저장할 total 변수를 0, 0부터 9까지 순차적으로 표시할 I 변수를 1로 초기화한다.
- ✓ **4~6행** i가 9 이하인 동안 반복문을 실행한다. 현재 i 값을 total에 더하고 i 값을 1씩 증가시킨다.
- ✓ **8행** 반복문이 종료되면 합계를 출력한다.

break와 continue란?

파이썬에서 break와 continue는 반복문을 제어하는 데 사용되는 두 가지 중요한 키워드이다. 이들은 반복문 내에서 실행 흐름을 변경하는 데 도움을 준다. break는 반복문을 완전히 종료시키는 데 사용되며, continue는 현재 반복을 건너뛰고 다음 반복으로 넘어가게 한다.

break

반복문 내에서 break를 만나면, 반복문이 즉시 종료되고 반복문 다음에 있는 코드가 실행된다.

break 사용 예시

```
1  count = 0
2  while True:
3      print("반복문을 %d번째 실행중입니다."%count)
4      if count == 5:
5          break
6      count += 1
```

- ✓ 1행~2행 count 변수를 0으로 초기화하고 무한 루프를 시작한다.
- ✓ 3행 현재 count 값을 포함한 문자열을 출력한다.
- ✓ **4행** count 값이 5와 같으면 break문에 의해 반복문이 종료된다.
- ✓ 5행 다음 반복으로 넘어가기 전에 count 변수를 1 증가시킨다.

continue

반복문 내에서 continue를 만나면, 현재 반복을 건너뛰고 반복문의 조건 확인 부분으로 돌아가 다음 반복을 시작한다.

continue 사용 예시

```
1  for i in range(1, 11):
2      if i % 2 != 0:
3          continue
4      print(i)
```

출력 결과

```
2
4
6
8
10
```

✓ 1행 for문으로 1부터 10까지 반복한다.

✓ **2~3행** 홀수인 경우 continue가 실행되어 현재 반복을 건너뛰고 다음 반복으로 넘어가게 한다. 이 경우 4행의 print(i)문은 실행되지 않는다.

✓ **4행** i가 짝수인 경우, 짝수인 i 값만 출력된다.

04.함수

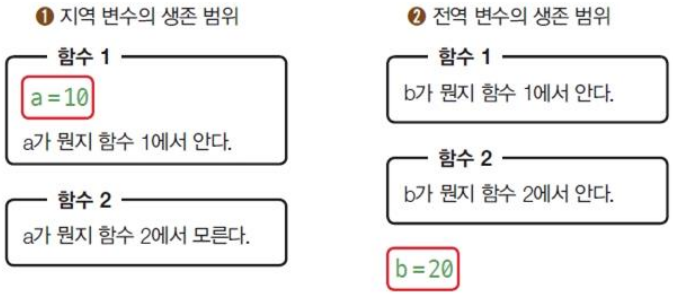
지역변수와 전역변수란?

전역 변수(Global Variable)

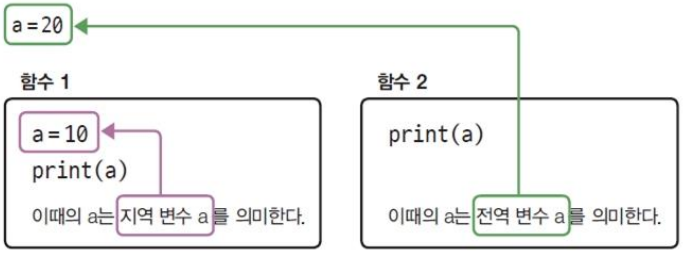
프로그램 전체 영역에서 접근 가능한 변수를 말한다.

지역 변수(Local Variable)

함수나 블록 내에서 선언되어 해당 함수나 블록 내에서만 접근 가능한 변수를 말한다.



지역 변수와 전역 변수의 생존 범위



지역 변수와 전역 변수의 공존

지역변수와 전역변수

```
1  ## 함수 선언 부분 ##
2  def func1():
3      a = 10  # 지역 변수
4      global b
5      b = "global variable"
6      print("func1()에서 a 값 %d" % a)
7
8  def func2():
9      print("func2()에서 a 값 %d" % a)
10     print("func2()에서 b 값 %s" % b)
11
12 ## 전역 변수 선언 부분 ##
13 a = 20  # 전역 변수
14
15 ## 메인 코드 부분 ##
16 func1()
17 func2()
```

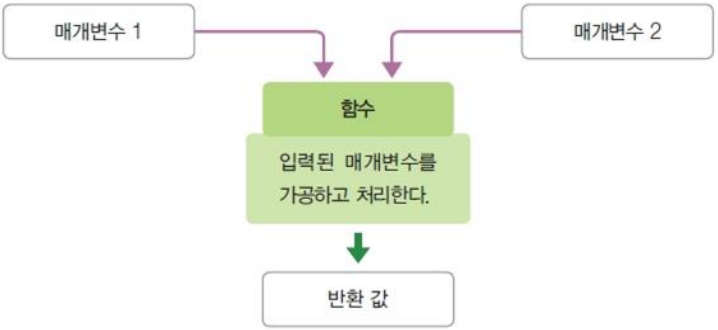
출력 결과

```
func1()에서 a 값: 10
func2()에서 a 값: 20
func2()에서 b 값: global variable
```

- ✓ **2~6행** func1() 함수를 정의하고 함수 내에서만 사용 가능한 지역 변수 a를 선언한다. global 키워드를 사용해서 함수 외부에서도 접근 가능한 전역 변수 b를 선언해 준다. 13행에서 전역변수 a가 선언되었지만 3행에서 지역변수 a를 선언하여 지역변수 a 값이 출력된다.
- ✓ **8~10행** 전역 변수 a와 b 값이 출력된다.
- ✓ **13행** 전역 변수 a를 선언한다. 이 변수는 함수 내부에서도 접근 가능하다.
- ✓ **16~17행** 위에서 선언한 func1(), func2() 함수를 호출한다.

함수란?

코딩을 하다 보면 같은 코드를 여러 번 동일하게 처리해야 할 때가 있다. 함수는 프로그램에서 반복되는 코드를 하나로 묶어서 필요할 때마다 호출할 수 있게 해주는 코드 블록이다. 함수를 사용하면 코드의 재사용성이 높아지고, 가독성이 좋아지며, 유지보수가 용이해진다.



함수는 매개변수를 입력받아 입력된 매개변수를 가공하고 처리하여 반환값을 돌려주는 역할을 한다. def는 함수를 만들 때 사용하는 예약어이며, 함수 이름은 함수를 만드는 사람이 임의로 정할 수 있다.

```
def 함수_이름(매개변수):
    수행할_문장1
    수행할_문장2
    ...
```

함수의 기본적인 구조

다음 페이지에서 실습을 통해 함수의 구조에 대해 알아보도록 하자

함수의 구조

```
1  def add(a, b): # a, b는 매개변수
2      return a+b
3
4  a = 3
5  b = 4
6  print(add(a, b)) # 3, 4는 인수
```

출력 결과

```
7
```

- ✓ **1~2행** add 함수를 정의하여 a와 b를 매개변수로 받아서 더한 값을 반환한다.
- ✓ **4~5행** 전역변수 a, b의 값을 할당한다.
- ✓ **6행** add 함수를 호출하여 전역변수 a, b의 값을 인수로 전달하고, 결과를 출력한다.

매개변수와 인수

- 매개변수(parameter): 함수에 입력으로 전달된 값을 받는 변수를 말한다.
- 인수(arguments): 함수를 호출할 때 전달하는 입력값을 말한다.

✓ 함수에는 매개변수와 인수를 이용한 형태 뿐만 아니라 다양한 형태가 있다. 다양한 형태의 함수들을 몇 가지 예시를 통해 살펴보도록 하자. 매개변수를 받지 않기 때문에 함수 호출 시 인수를 전달할 필요가 없다.

입력 값이 없는 함수

```
1  def say():
2      return 'Hello'
3
4  print(say())
```

출력 결과

```
Hello
```

✓ **1행** say라는 이름의 매개변수를 받지 않는 함수를 정의한다. 함수가 호출되면 문자열 Hello를 반환한다.

✓ **4행** say() 함수를 호출한다.

리턴 값이 없는 함수

```
1  def add(a, b):
2      print("%d, %d의 합은 %d입니다." % (a, b, a+b))
3
4  add(2, 3)
```

출력 결과

```
2, 3의 합은 5입니다.
```

✓ **1~2행** add라는 이름의 a와 b를 매개변수로 받는 함수를 정의한다. 함수가 호출되면 내부의 print구문이 실행된다.

✓ **4행** add 함수를 호출하여 인수로 2와 3을 전달한다.

입력 값, 리턴 값이 없는 함수

```
1  def say():
2      print('Hello' )
3
4  say()
```

출력 결과

```
Hello
```

✓ **1~2행** say라는 매개변수를 받지 않는 함수를 정의한다. 함수가 호출되면 Hello 문자열을 출력한다.

✓ **4행** say 함수를 호출한다.

매개변수를 지정하여 호출하는 함수

```
1  def sub(a, b):
2      return a - b
3
4  result = sub(b=4, a=2)
5  print(result)
```

출력 결과

```
-2
```

✓ **1~2행** sub라는 이름의 a와 b를 매개변수로 받는 함수를 정의한다. 함수가 호출되면 a-b 값이 반환된다.

✓ **4행** sub 함수를 호출하면서 키워드 인수로 b=4, a=2를 전달한다. 이 때 키워드 인수의 순서는 지키지 않아도 된다. 함수 호출 시 반환된 값은 변수 result에 저장된다.

✓ **5행** 변수 result의 값을 출력한다.

여러 개의 입력 값을 받는 함수

```
1  def add_many(*args):
2      result = 0
3      for i in args:
4          result = result + i
5      return result
6
7  result = add_many(1,2,3,4,5)
8  print(result)
```

출력 결과

```
15
```

✓ **1~2행** *args를 매개변수로 받는 add_many 함수를 정의한다. *args는 가변 매개변수로, 여러 개의 인수를 받을 수 있다. 전달된 인수들은 튜플 형태로 args에 저장된다.

✓ **3~4행** args에 저장된 인수를 순서대로 변수 i에 대입하여 result에 저장한다.

✓ 5행 모든 인수의 합을 구한 후, result의 값을 반환한다.

✓ **7행** add_many 함수를 호출하면서 인수로 1, 2, 3, 4, 5를 전달한다.

✓ 8행 변수 result의 값을 출력한다.

키워드 매개변수, kwargs

```
1  def print_kwargs(**kwargs):
2      print(kwargs)
3
4  print_kwargs(name='Gildong', age=20)
```

출력 결과

```
{'name': 'Gildong', 'age': 20}
```

- ✓ **1행** **kwargs를 매개변수로 받는 print_kwargs 함수를 정의한다. **kwargs는 가변 키워드 매개변수로, 개수에 상관 없이 여러 개의 키워드 인수를 받을 수 있다. 전달된 키워드 인수들은 딕셔너리 형태로 kwargs에 저장된다.
- ✓ **2행** 함수 호출 시 kwargs에 저장된 값을 출력한다.
- ✓ **4행** print_kwargs 함수를 호출해 키워드 인수로 name=Gildong, age=20을 전달한다.

함수의 장점

- 재사용성: 한 번 정의한 함수는 여러 번 사용이 가능하다.
- 가독성: 코드를 함수로 묶으면 전체 코드의 구조가 명확해지고 읽기 쉬워진다.
- 유지보수 용이성: 함수 내부의 코드가 변경되어야 할 때, 함수만 수정하면 되므로 유지보수가 용이하다.

05. 예외 처리

예외 처리란?

코딩을 하다 보면 예상치 못한 오류가 발생할 수 있다. 이러한 오류를 예외라고 하며, 예외가 발생하면 프로그램이 중단될 수 있다. 파이썬은 이러한 예외를 처리하여 프로그램 중단 없이 정상적으로 동작하도록 도와주는 예외 처리 기능을 제공한다.

예외의 종류

- SyntaxError: 문법이 잘못되었을 경우 예외 발생
- ZeroDivisionError: 0으로 나누기 연산을 시도하면 예외 발생
- TypeError: 서로 다른 타입 간 연산을 하려고 할 때 예외 발생
- KeyError: 딕셔너리에서 존재하지 않는 키를 참조할 때 예외 발생
- ValueError: 연산이나 함수에 부적절한 값을 전달할 때 예외 발생
- IndexError: 리스트나 튜플 등의 인덱스가 범위를 벗어나면 예외 발생
- FileNotFoundError: 존재하지 않는 파일이나 디렉토리에 접근하면 예외 발생

예외 처리의 기본 구조

파이썬에서 예외 처리는 try, except, else, finally 블록으로 수행 가능하다.

예외 처리의 예시

```
1  try:
2      result = 10 / 0
3  except ZeroDivisionError:
4      print("0으로 나눌 수 없다.")
5  else:
6      print("나눗셈이 성공적으로 수행되었습니다.")
7  finally:
8      print("예외 처리 완료.")
```

출력 결과

```
0으로 나눌 수 없다.
예외 처리 완료.
```

- ✓ **1행** try 블록 내부 코드가 실행되며, 예외가 발생하면 해당 예외에 대응하는 except 블록이 실행된다.
- ✓ **2행** 0으로 나누기 연산은 ZeroDivisionError 예외를 발생시킨다.
- ✓ **3~4행** ZeroDivisionError가 발생했기 때문에 해당 블록이 실행된다.
- ✓ **4~5행** try 블록 내부 코드에서 예외가 발생하지 않은 경우 실행된다.
- ✓ **7~8행** 예외 발생 여부와 상관없이 항상 실행되는 코드이다.

제 3 장

리스트

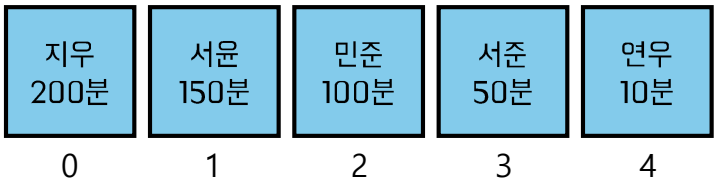
1) 선형 리스트
2) 단순 연결 리스트
3) 원형 리스트

01. 선형 리스트

선형 리스트

선형 리스트(Linear List)는 데이터를 일정한 순서로 나열한 자료구조이다. 다른 이름으로는 순차 리스트(Ordered List)라고도 불린다. 선형 리스트의 특징으로는 메모리가 연속적이며, 입력 순서대로 저장하는 데이터에 사용하기 적합한 자료구조이다.

한가지 예시로 연구실 학생들 중 공부를 한 시간이 긴 순서대로 선형 리스트에 저장한다고 가정해 보자.



데이터 삽입

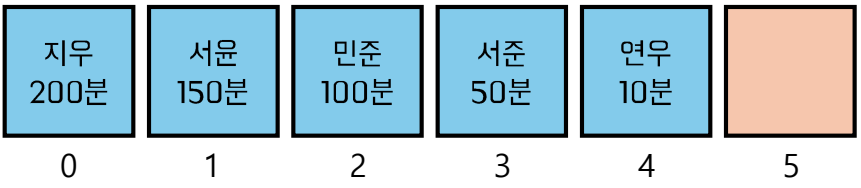
이때 연구실에 새로 들어온 하윤이가 공부를 60분 했다면 민준이와 서준이 사이에 하윤이를 삽입 해주어야 한다. 하지만 민준이와 서준이 사이에는 빈 칸이 없기 때문에 몇가지 단계를 거쳐야 한다.

1. 맨 끝에 새로운 빈칸을 생성
2. 데이터를 삽입할 칸 까지 한 칸 씩 뒤로 옮김
3. 확보한 빈 칸에 데이터 삽입

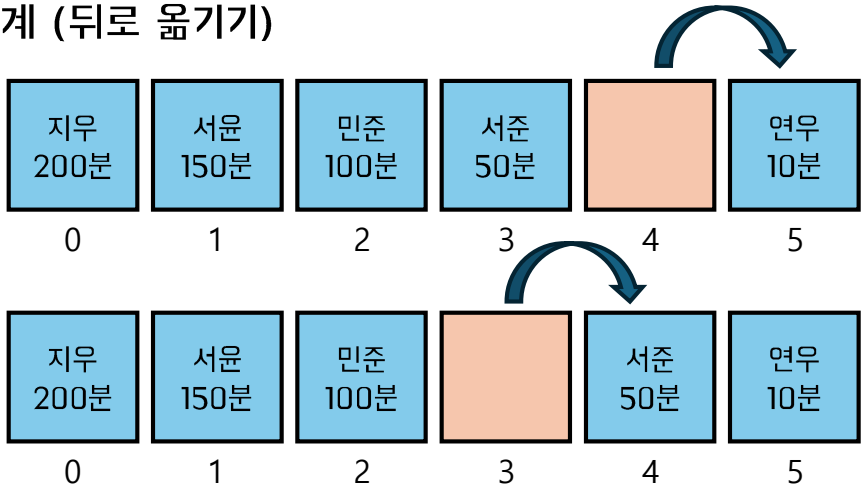
데이터 삽입

위 단계를 그림으로 표현하면 다음과 같다.

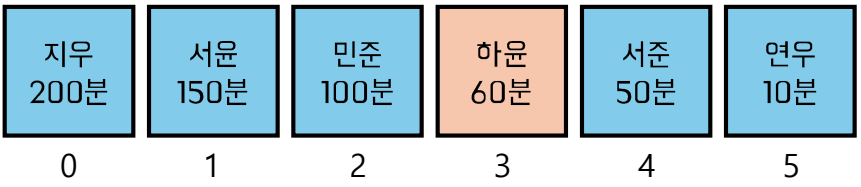
1단계 (빈 칸 생성)



2단계 (뒤로 옮기기)



3단계 (데이터 삽입)



데이터 삽입

데이터 삽입을 의사 코드 형태로 나타내면 다음과 같다.

3-1. 데이터 삽입

```
study.append(None)
for 현재 위치 in range(마지막 위치, 지정 위치, -1):
    study[현재 위치] = study[현재 위치 - 1]
    study[현재 위치 - 1] = None
study[지정 위치] = student
```

데이터 삭제

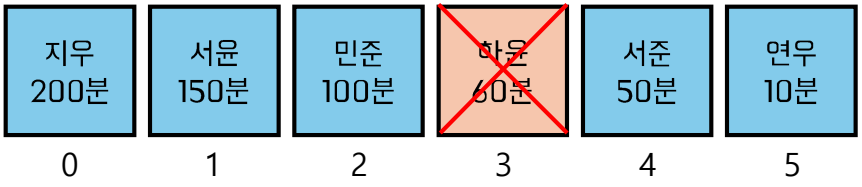
이번에는 데이터가 선형 리스트의 중간 부분에서 삭제되는 경우를 생각해 보도록 하자. 하윤이가 연구실에서 탈퇴한다면 선형 리스트에서 하윤이를 삭제해 주어야 한다. 이때, 중간 부분에 위치한 하윤이를 삭제하면 중간 부분의 칸이 비어 있게 되기 때문에 이번에도 몇가지 단계를 거쳐 데이터를 삭제해야 한다.

1. 데이터 삭제
2. 데이터를 삭제하고 비어 있는 칸의 뒤에 있는 데이터를 앞으로 한 칸 씩 옮김
3. 빈칸 삭제

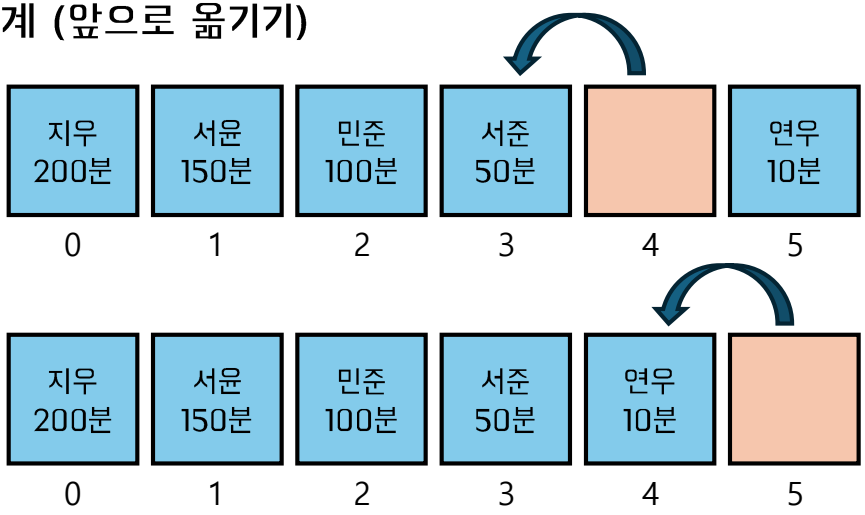
데이터 삭제

위 단계를 그림으로 표현하면 다음과 같다.

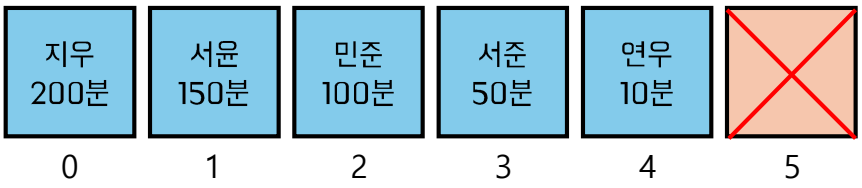
1단계 (데이터 삭제)



2단계 (앞으로 옮기기)



3단계 (빈 칸 삭제)



데이터 삭제

데이터 삭제를 의사 코드 형태로 나타내면 다음과 같다.

3-2. 데이터 삭제

```
study[지정 위치]=None
for 현재 위치 in range(지정 위치 + 1, 마지막 위치 + 1):
    study[현재 위치 - 1] = study[현재 위치]
    study[현재 위치] = None
del(study[지정 위치])
```

선형 리스트 구현

지금까지 선형 리스트의 구조와 동작에 대해서 알아보았고, 여기서는 사용자가 직접 데이터를 입력해서 가변적으로 사용할 수 있는 선형 리스트를 구현하는 코드를 구현해보도록 하자.

먼저 선형 리스트를 생성하고 맨 뒤 칸에 데이터를 추가하는 함수를 만들어 보도록 하겠다.

3-3. 선형 리스트 생성

```
1  study = []
2
3  def add_data(student) :
4      study.append(None)
5      SLen = len(study)
6      study[SLen - 1] = student
```

✓ 1행 : 빈 배열을 준비한다.

✓ 3~6행 : 배열의 맨 뒤에 데이터를 추가하는 함수로, 매개변수로는 저장할 데이터가 필요하다. 맨 뒤에 빈 칸을 추가 하고 배열의 크기를 구한 뒤, 맨 뒤에 매개변수로 받은 데이터를 추가한다.

선형 리스트 구현

이번에는 생성되어 있는 선형 리스트에 데이터를 삽입하는 함수를 만들어 보도록 하겠다.

3-4. 데이터 삽입

```
1   study = ["지우","서윤","민준","서준","연우"]
2
3   def insert_data(position, student) :
4       if position < 0 or position > len(study):
5           print("데이터를 삽입할 범위를 벗어났습니다.")
6           return
7
8       study.append(None)
9       SLen = len(study)
10      for i in range(SLen - 1, position, -1):
11          study[i] = study[i - 1]
12          study[i - 1] = None
13      study[position] = student
```

✓ 1행 : 배열에 데이터를 미리 넣어 놓는다.

✓ 3행 : 데이터를 삽입하는 함수를 선언한다.

✓ 4~6행 : 데이터를 삽입할 위치가 배열 범위를 벗어났을 경우 메시지를 출력하고 함수를 종료한다.

✓ 8~13행 : 맨 뒤에 빈 칸을 생성하고 데이터를 삽입할 위치의 데이터까지 뒤로 한 칸 씩 옮긴 다음 해당 위치에 데이터를 삽입한다.

선형 리스트 구현

이번에는 생성되어 있는 선형 리스트에 데이터를 삭제하는 함수를 만들어 보도록 하겠다.

3-5. 데이터 삭제

```
1   study = ["지우","서윤","민준","서준","연우"]
2
3   def delete_data(position) :
4       if position < 0 or position > len(study):
5           print("데이터를 삭제할 범위를 벗어났습니다.")
6           return
7
8       study[position] = None
9       SLen = len(study)
10      for i in range(position + 1, SLen):
11          study[i - 1] = study[i]
12          study[i] = None
13      del(study[SLen - 1])
```

✓ 1행 : 배열에 데이터를 미리 넣어 놓는다.

✓ 3행 : 데이터를 삭제하는 함수를 선언한다.

✓ 4~6행 : 데이터를 삭제할 위치가 배열 범위를 벗어났을 경우 메시지를 출력하고 함수를 종료한다.

✓ 8~13행 : 해당 위치의 칸을 빈 칸으로 만들고 해당 위치 까지의 데이터를 앞으로 한 칸 씩 옮긴 뒤, 맨 뒤의 빈 칸을 삭제한다.

선형 리스트 구현

앞에서 만들어 본 코드를 사용해 선형 리스트를 작업하는 코드를 만들어 보도록 하겠다.

3-6. 선형 리스트 작업

```
study = []
select = -1

if __name__ == "__main__" :
    while (select != 4):
        select = int(input("선택하세요(1:추가, 2:삽입, 3:삭제, 4:종료)-->"))
        if (select == 1):
            data = input("추가할 데이터-->")
            add_data(data)
            print(study)
        elif (select == 2):
            pos = int(input("삽입할 위치-->"))
            data = input("추가할 데이터-->")
            insert_data(pos, data)
            print(study)
        elif (select == 3):
            pos = int(input("삭제할 위치-->"))
            delete_data(pos)
            print(study)
        elif (select == 4):
            print(study)
            exit

        else:
            print("1~4 중 하나를 입력하세요.")
            continue
```

선형 리스트 구현

- ✓ 5행 : 사용자가 4를 입력할 때 까지 계속해서 반복한다.
- ✓ 6행 : 어떤 작업을 수행할지 입력 받는다.
- ✓ 7~19행 : 입력 받은 숫자에 해당하는 함수를 호출하여 추가, 삽입, 삭제 작업을 수행한다.
- ✓ 20~22행 : 배열을 출력하고 작업을 종료한다.
- ✓ 24~26행 : 잘못된 숫자를 입력한 경우 안내 문구를 출력한다.

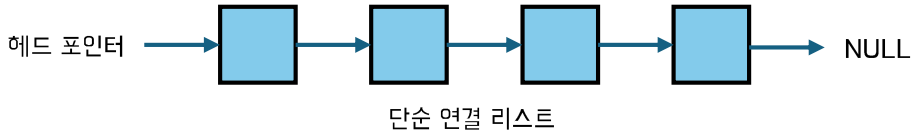
02. 단순 연결 리스트

연결 리스트

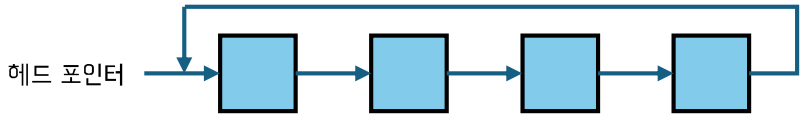
연결 리스트는 리스트를 만드는 방법 중 하나이다. 연결 리스트를 이용해서 리스트를 구현하는 방법은 구현하는 것이 복잡하고 오류가 발생하기 쉽다는 단점이 있지만, 삽입과 삭제가 효율적이고 배열 처럼 크기가 제한되지 않는다는 장점이 있다.

연결 리스트를 구성하는 요소로는 노드(Node)와 헤드 포인터(Head pointer)가 있다. 노드는 데이터 값을 저장하고 있는 데이터 필드와 다음 노드를 가리키는 링크 필드로 구성되어 있다. 헤드 포인터는 연결 리스트의 첫번째 노드를 가리키는 포인터 변수이다.

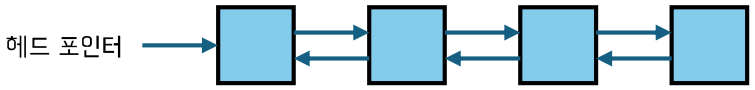
연결 리스트의 종류로는 단순 연결 리스트, 원형 연결 리스트, 이중 연결 리스트가 있다. 이번 장에서는 단순 연결 리스트와 원형 연결 리스트를 알아보도록 하겠다.



단순 연결 리스트



원형 연결 리스트

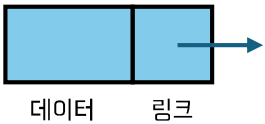


이중 연결 리스트

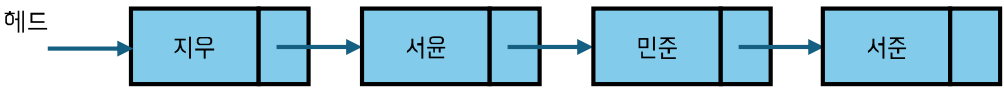
단순 연결 리스트

단순 연결 리스트(Singly Linked List)는 하나의 링크 필드를 이용하여 연결하는 방식으로 마지막 노드의 링크 값에는 NULL 값이 들어간다. 원형 연결 리스트와 이중 연결 리스트와는 다르게 한 방향으로만 연결되어 있기 때문에 단일 연결 리스트라고도 말할 수 있다. 앞서 살펴본 선형 리스트와 다르게 삽입 또는 삭제 연산을 할 때에 연결 링크만 조정해주면 끝이기 때문에 메모리상의 움직임 없이 작업할 수 있다는 장점이 있다.

앞서 간단하게 설명했던 노드의 구조를 그림으로 표현하면 다음과 같다.



따라서, 단순 연결 리스트를 그림으로 명확하게 표현해보면 다음과 같이 나타낼 수 있다.

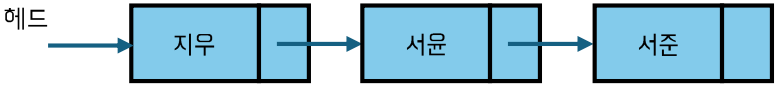


단순 연결 리스트

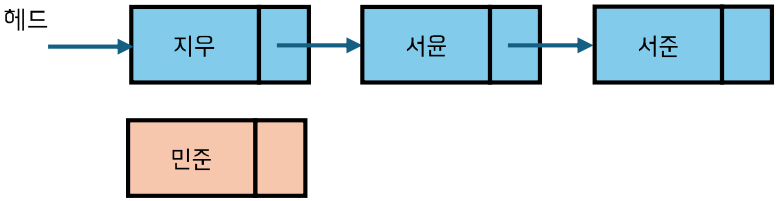
이번에는 단순 연결 리스트에서 데이터를 삽입하거나 삭제하는 과정을 그림으로 살펴보도록 하자. 선형 리스트에서 데이터를 삽입하거나 삭제하는 것과 다르게 빈 칸이 생기지 않고 더욱 간단하게 구현할 수 있다는 특징이 있다.

노드 삽입 (데이터 삽입)

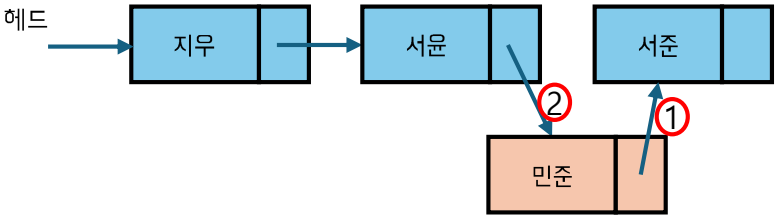
다음과 같은 단순 연결 리스트가 있는 상황에서 민준이를 서윤이와 서준이 사이에 삽입하는 과정을 고려해 보자.



1단계 (삽입할 노드 생성)



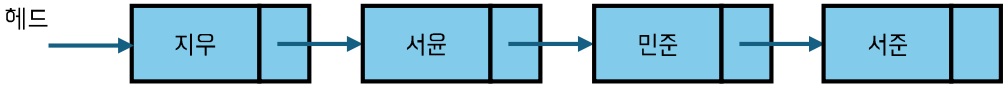
2단계 (링크 수정)



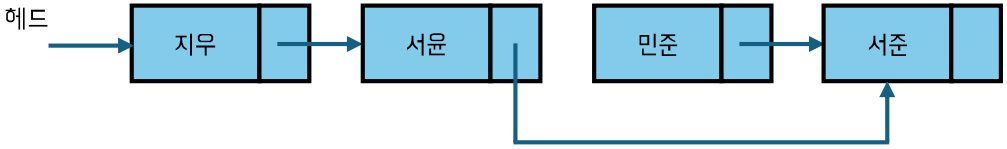
단순 연결 리스트

노드 삭제 (데이터 삭제)

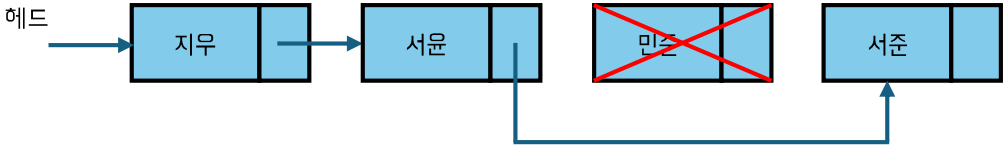
이번에는 다음과 같은 단순 연결 리스트에서 민준이를 삭제하는 상황을 고려해 보도록 하자.



1단계 (링크 수정)



2단계 (노드 삭제)



단순 연결 리스트 구현

단순 연결 리스트를 생성하는 코드를 작성하기에 앞서 헤드(Head), 현재(Current), 이전(Pre)에 대해 간략하게 설명하겠다. Head는 항상 첫 번째 노드를 가리키고 있어야 한다. Current는 현재 처리하고 있는 노드를 가리키고, Pre는 Current가 가리키고 있는 노드의 바로 앞 노드를 가리키고 있는 변수이다. 이 3개의 변수를 선언하여 None으로 초기화 해주는 과정이 필요하다.

이제 배열에 저장되어 있는 데이터를 가져와 단순 연결 리스트를 생성하는 방법을 알아보자. 먼저, 첫 번째 노드를 생성하는 방법이다.

1. 빈 노드 생성
2. 데이터 가져오기
3. 해당 노드를 head로 지정
4. 메모리에 노드 저장

두 번째 부터의 노드들을 생성하는 방법은 다음과 같다.

1. 첫 번째 노드를 pre로 지정
2. 빈 노드 생성
3. 데이터 가져오기
4. pre의 링크를 새로 생성한 노드에 연결
5. 메모리에 새로 생성한 노드 저장

단순 연결 리스트 구현

앞서 설명한 방법을 활용하여 단순 연결 리스트를 생성하는 코드를 작성해 보도록 하겠다.

3-7. 단순 연결 리스트 생성

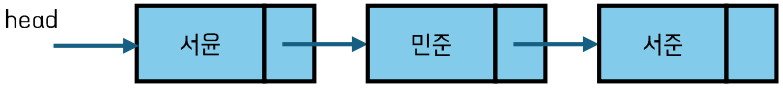
```
Class Node() :
    def __init__(self):
        self.data = None
        self.link = None

memory = []
head, current, pre = None, None, None

dataarray = ["지우","서윤","민준","서준"]

if __name__ == "__main__" :
    node = Node()
    node.data = dataarray[0]
    head = node
    memory.append(node)

    for data in dataarray[1:] :
        pre = node
        node = Node()
        node.data = data
        pre.link = node
        memory.append(node)
```

단순 연결 리스트

- ✓ 1~4행 : 노드의 데이터와 링크를 위한 노드 클래스를 정의한다.
- ✓ 6~7행 : 노드를 저장할 메모리와 head, current, pre를 선언하고 초기화 해준다.
- ✓ 9행 : 단순 연결 리스를 생성할 데이터를 배열에 저장한다.
- ✓ 12~15행 : 빈 노드를 생성하고 배열의 첫 번째 데이터를 가져온다. 해당 노드를 head로 지정하고 메모리에 저장한다.
- ✓ 17~22행 : 현재 노드를 pre로 지정하고 빈 노드를 생성한다. 배열의 데이터를 가져오고 pre의 링크를 해당 노드에 연결한 뒤 메모리에 저장한다.

단순 연결 리스트

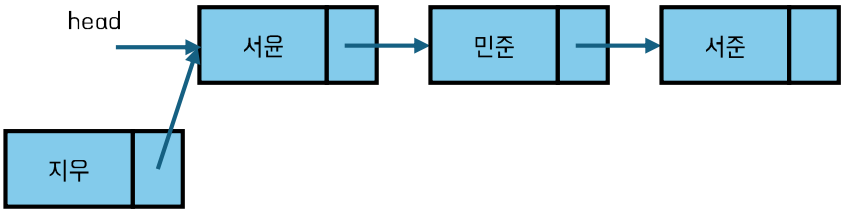
이번에는 앞에서 생성한 단순 연결 리스트에 노드를 삽입하는 방법을 알아보도록 하자. 먼저 첫 번째 노드를 삽입하는 방법이다.



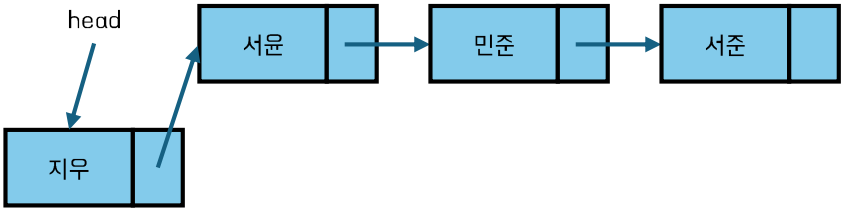
위와 같은 단순 연결 리스트가 있는 상황에서 첫 번째 노드에 지우를 삽입하는 상황을 고려해 보자.

1. 새 노드 생성

2. 새 노드의 링크를 첫 번째 노드(head가 가리키고 있는 노드)로 연결

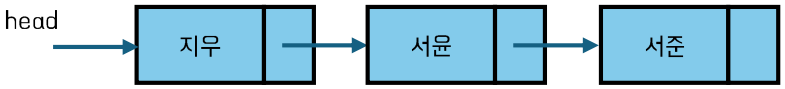


3. head를 새 노드로 연결

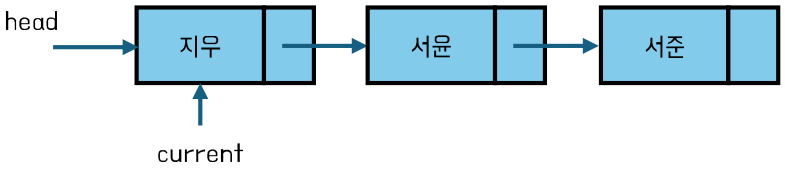


단순 연결 리스트

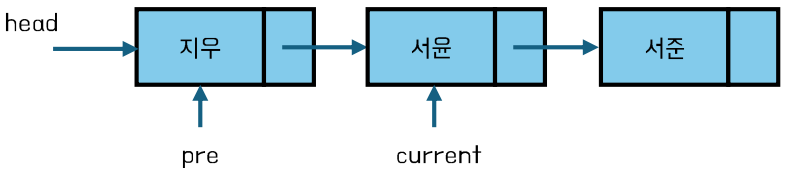
이번에는 아래와 같은 단순 연결 리스트가 있을 때, 민준이를 서윤이와 서준이 사이에 삽입하는 상황을 고려해 보도록 하자.



1. head노드를 current노드로 지정하고 current노드가 서준이 인지 확인

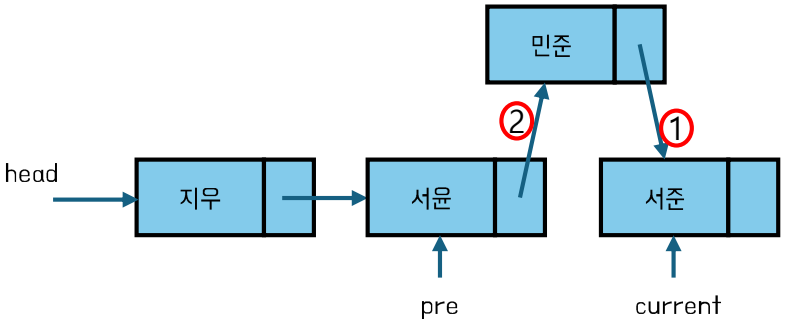


2. current노드를 pre노드로 지정하고 다음 노드를 current노드로 지정 current노드가 서준이 인지 확인, current노드가 서준이가 될 때 까지 이 단계를 반복



단순 연결 리스트

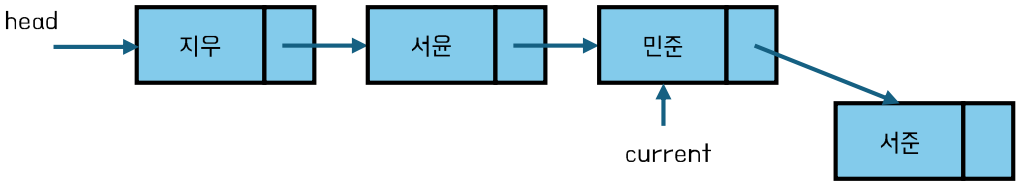
3. Current노드가 서준이라면 새 노드 생성 후 current 노드에 링크 연결
   pre노드의 링크를 새 노드에 연결



이번에는 같은 상황에서 마지막에 노드를 삽입 하는 상황을 고려해보도록 하자.

마지막에 노드를 삽입하려면 단순 연결 리스트에 없는 노드를 찾는다.

1. 중간에 노드를 삽입하는 과정과 동일하게 수행
2. 마지막 노드까지 확인했을 때 까지 찾고자 하는 노드가 없다면 새 노드를 생성한 뒤 current노드의 링크를 새 노드에 연결



단순 연결 리스트

앞에서 설명한 첫 번째 노드, 중간 노드, 마지막 노드 삽입을 모두 고려해서 노드를 삽입하는 코드를 작성해 보도록 하겠다.

3-8. 단순 연결 리스트 노드 삽입

```
1  def insertNode(findData, insertData) :
2      global memory, head, current, pre
3
4      if head.data == findData :
5          node = Node()
6          node.data = insertData
7          node.link = head
8          head = node
9          return
10
11     current = head
12     while current.link != None :
13         pre = current
14         current = current.link
15         if current.data == findData :
16             node = Node()
17             node.data = insertData
18             node.link = current
19             pre.link = node
20             return
21     node = Node()
22     node.data = insertData
23     current.link = node
```

단순 연결 리스트

- ✓ 1행 : 노드를 삽입하는 함수를 선언한다. 찾을 데이터와 삽입할 데이터를 매개변수로 받는다.
- ✓ 4~9행 : head의 데이터가 찾는 데이터이면(첫 번째 노드에 삽입하는 경우) 빈 노드를 생성하고 삽입할 데이터를 넣어주고 기존 head노드에 링크를 연결하고 head가 해당 노드를 가리키게 한다.
- ✓ 11~20행 : head노드를 current로 지정하고 링크가 끊길 때 까지 반복한다. pre를 current로 지정하고 current는 다음 노드로 바꿔준다. current의 데이터가 찾는 데이터라면 빈 노드를 생성하고 삽입할 데이터를 넣어주고 current노드에 링크를 연결하고 pre노드가 해당 노드를 가리키게 한다.
- ✓ 21~23행 : 마지막 까지 찾는 데이터를 못 찾았다면 빈 노드를 생성하고 삽입할 데이터를 넣어주고 current노드가 해당 노드를 가리키게 한다.

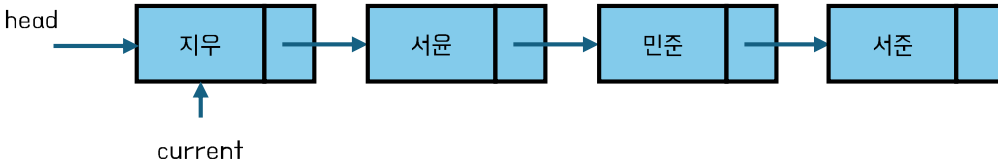
단순 연결 리스트

이번에는 단순 연결 리스트에서 노드를 삭제하는 방법에 대해서 알아보도록 하자.

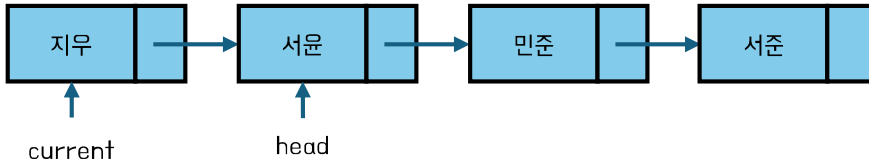
먼저 첫 번째 노드를 삭제하는 방법이다.

위와 같은 단순 연결 리스트에서 지우를 삭제하는 상황을 고려해보도록 하자.

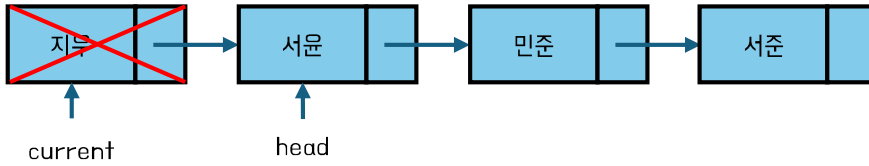
1. current노드를 head노드로 지정



2. head노드를 head노드가 가리키던 노드로 변경



3. current노드 삭제

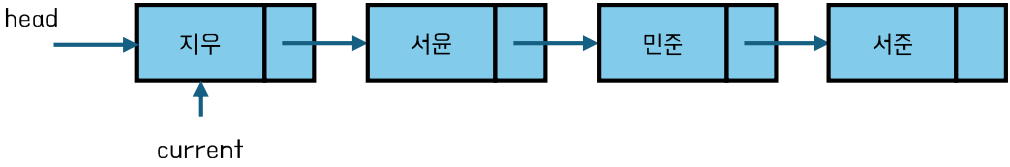


단순 연결 리스트

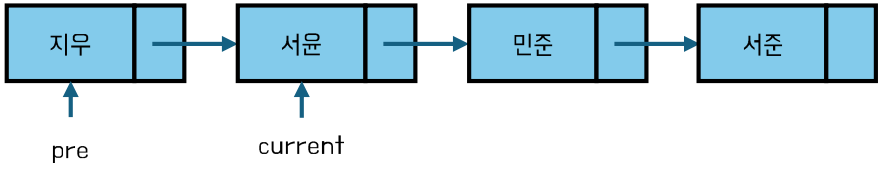
이번에는 첫 번째가 아닌 노드를 삭제하는 방법에 대해서 알아보도록 하자.

마찬가지로 다음과 같은 단순 연결 리스트에서 민준이를 삭제하는 상황을 고려해보도록 하자.

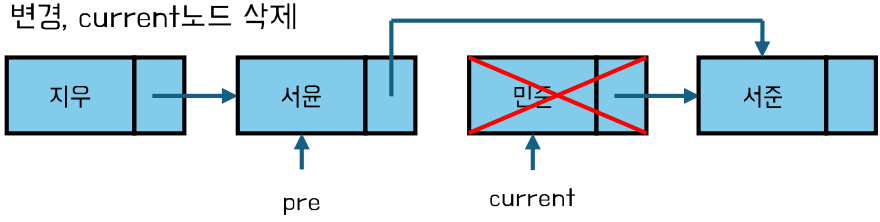
1. current를 head노드로 지정하고 민준이 인지 확인



2. pre를 current노드로 지정하고 current를 다음 노드로 변경한 뒤 민준이 인지 확인, current노드가 민준이가 될 때 까지 반복



3. current노드가 민준이라면 pre노드의 링크를 current노드의 링크가 가리키는 노드로 변경, current노드 삭제



단순 연결 리스트

앞에서 설명한 상황을 고려해서 노드를 삭제하는 코드를 작성해보도록 하겠다.

3-9. 단순 연결 리스트 노드 삭제

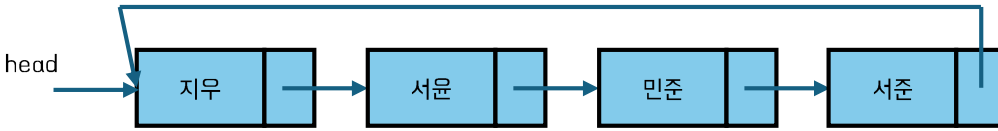
```
1  def deleteNode(deleteData) :
2      global memory, head, current, pre
3
4      if head.data == deleteData :
5          current = head
6          head = head.link
7          del(current)
8          return
9
10     current = head
11     while current.link != None :
12         pre = current
13         current = current.link
14         if current.data == deleteData :
15             pre.link = current.link
16             del(current)
17             return
```

- ✓ 1행 : 노드를 삭제하는 함수를 선언한다.
- ✓ 4~8행 : head의 데이터가 삭제할 데이터인 경우 current가 head를 가리키도록 하고 head는 head의 링크가 가리키는 노드로 변경한 뒤 current 노드를 삭제한다.
- ✓ 10~17행 : current가 head를 가리키도록 하고 current의 링크가 끊길 때 까지 반복한다. pre를 current를 가리키도록 하고 current는 다음 노드로 이동한다. current노드의 데이터가 삭제할 데이터라면 pre의 링크를 current의 링크가 가리키는 노드로 변경하고 current를 삭제한다.

03. 원형 연결 리스트

원형 연결 리스트

원형 연결 리스트(Circular Linked List)는 앞서 알아본 단순 연결 리스트와 구조와 작동 원리, 구현 방법이 매우 비슷하다. 다만, 단순 연결 리스트는 마지막 노드까지 방문 했다면 프로그램이 종료되기 때문에 다시 방문하려면 첫 번째 노드부터 다시 시작해야 하는 반면, 원형 연결 리스트는 마지막 노드가 다시 첫 번째 노드를 가리키기 때문에 계속해서 방문이 가능하다.



원형 연결 리스트의 경우 노드를 생성하고 삽입, 삭제하는 원리는 앞서 단순 연결 리스트와 매우 유사하기 때문에 이에 대한 설명은 생략하고 원형 연결 리스트를 구현해 보도록 하겠다. 다만, 원형 연결 리스트의 경우 마지막 노드가 첫 번째 노드를 가리켜야 하기 때문에 이 부분에 유의해서 구현해야 한다.

원형 연결 리스트 구현

원형 연결 리스트를 생성하는 코드를 작성해 보도록 하겠다.

3-10. 원형 연결 리스트 생성

```
Class Node() :
    def __init__(self):
        self.data = None
        self.link = None

memory = []
head, current, pre = None, None, None

dataarray = ["지우","서윤","민준","서준"]

if __name__ == "__main__" :
    node = Node()
    node.data = dataarray[0]
    head = node
    node.link = head
    memory.append(node)

    for data in dataarray[1:] :
        pre = node
        node = Node()
        node.data = data
        pre.link = node
        node.link = head
        memory.append(node)
```

원형 연결 리스트 구현

- ✓ 1~4행 : 노드의 데이터와 링크를 위한 노드 클래스를 정의한다.
- ✓ 6~7행 : 노드를 저장할 메모리와 head, current, pre를 선언하고 초기화한다.
- ✓ 9행 : 원형 연결 리스트를 생성할 데이터를 배열에 저장한다.
- ✓ 12~16행 : 빈 노드를 생성하고 배열의 첫 번째 데이터를 가져온다. 해당 노드를 head로 지정한다. 15행에서 해당 노드의 링크를 head로 연결하는데 이는 자기 자신을 가리키는 형태가 된다. 이 부분이 단순 연결 리스트와의 차이점이다.
- ✓ 18~24행 : 현재 노드를 pre로 지정하고 빈 노드를 생성한다. 배열의 데이터를 가져오고 pre의 링크를 해당 노드에 연결하고 해당 노드의 링크를 head 노드에 연결한다. 이 부분이 단순 연결 리스트와의 차이점이다.

원형 연결 리스트 구현

이번에는 원형 연결 리스트에 노드를 삽입하는 코드를 구현해 보도록 하겠다.

노드를 삽입하는 방법은 단순 연결 리스트와 매우 비슷하지만 항상 마지막 노드가 첫 번째 노드를 가리켜야 한다는 점에 유의해야 한다.

3-11. 원형 연결 리스트 노드 삽입

```
1  def insertNode(findData, insertData) :
2      global memory, head, current, pre
3
4      if head.data == findData :
5          node = Node()
6          node.data = insertData
7          node.link = head
8          last = head
9          while last.link != head :
10             last = last.link
11         last.link = node
12         head = node
13         return
14
15     current = head
16     while current.link != head :
17         pre = current
18         current = current.link
19         if current.data == findData :
20             node = Node()
21             node.data = insertData
22             node.link = current
23             pre.link = node
24             return
25     node = Node()
26     node.data = insertData
27     current.link = node
28     node.link = head
```

원형 연결 리스트 구현

- ✓ 1행 : 노드를 삽입하는 함수를 선언한다.
- ✓ 4~13행 : head의 데이터가 찾는 데이터라면 새 노드를 생성하고 데이터를 넣어준 뒤 head에 링크를 연결한다. last라는 변수를 생성하여 head를 가리키도록 초기화 해준 뒤 last의 링크가 head가 아닌 동안 한 칸 씩 이동하여 마지막 노드를 찾는다. 마지막 노드를 찾았다면 마지막 노드의 링크를 해당 노드에 연결하고 head도 해당 노드에 연결한다.
- ✓ 15~24행 : current의 링크가 head를 가리키지 않는 동안(마지막 노드 전)까지만 반복한다. pre가 current 노드를 가리키게 하고 current는 한 칸 뒤로 이동한다. current의 데이터가 찾는 데이터라면 새 노드를 생성하고 데이터를 넣어준 뒤 current노드에 링크를 연결하고 pre노드는 해당 노드에 링크를 연결한다.
- ✓ 25~28행 : 마지막 까지 찾는 데이터가 없었을 경우 새 노드를 생성하고 데이터를 넣어준 뒤 current노드의 링크를 해당 노드에 연결하고 해당 노드의 링크를 head노드에 연결한다.

원형 연결 리스트 구현

이번에는 원형 연결 리스트에 노드를 삭제하는 코드를 구현해 보도록 하겠다.

노드를 삭제하는 방법은 단순 연결 리스트와 매우 비슷하지만 항상 마지막 노드가 첫 번째 노드를 가리켜야 한다는 점에 유의해야 한다.

3-12. 원형 연결 리스트 노드 삭제

```
1  def deleteNode(deleteData) :
2      global memory, head, current, pre
3
4      if head.data == deleteData :
5          current = head
6          head = head.link
7          last = head
8          while last.link != current :
9              last = last.link
10         last.link = head
11         del(current)
12         return
13
14     current = head
15     while current.link != head :
16         pre = current
17         current = current.link
18         if current.data == deleteData :
19             pre.link = current.link
20             del(current)
21             return
```

원형 연결 리스트 구현

- ✓ 1행 : 노드를 삭제하는 함수를 선언한다.
- ✓ 4~12행 : head의 데이터가 삭제할 데이터라면 current가 head를 가리키도록 하고 head를 다음 노드로 이동한다. last변수를 새로 만들어 head를 가리키도록 하고 last노드가 마지막 노드가 될 때 까지 이동한 뒤 last노드가 head를 가리키도록 하고 current노드를 삭제한다.
- ✓ 14~21행 : current가 head를 가리키고 current노드의 링크가 head를 가리키지 않을 때 까지 반복한다. pre가 current노드를 가리키고 current는 다음 노드로 이동한다. current노드의 데이터가 삭제할 데이터라면 pre노드의 링크를 current노드의 링크 노드와 연결하고 current노드를 삭제한다.

제 4 장

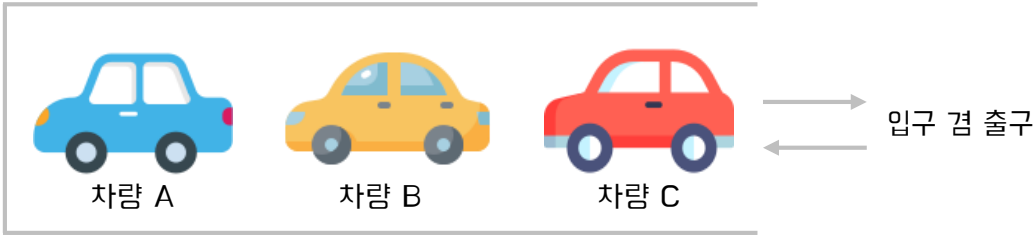
스택/큐

1) 스택
2) 큐

01. 스택이란?

스택 구조란?

스택(stack)구조란 입구와 출구가 한 개인 주차장처럼 가장 먼저 들어간 차량이 가장 마지막에 나올 수 있는 구조를 말한다. (선입후출, 후입선출)



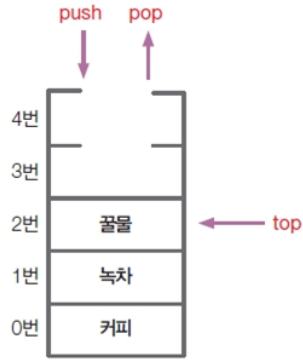
위 그림에 있는 차량은 A, B, C 순으로 들어갔으며 입구 겸 출구가 한 곳이기 때문에 다시 나오기 위해서는 C, B, A 순서로 나와야 한다.

스택의 실생활 사례

- 프링글스 통
- 종이컵 수거함
- 함수 호출 스택
- 웹 브라우저의 뒤로 가기 버튼

스택이란?

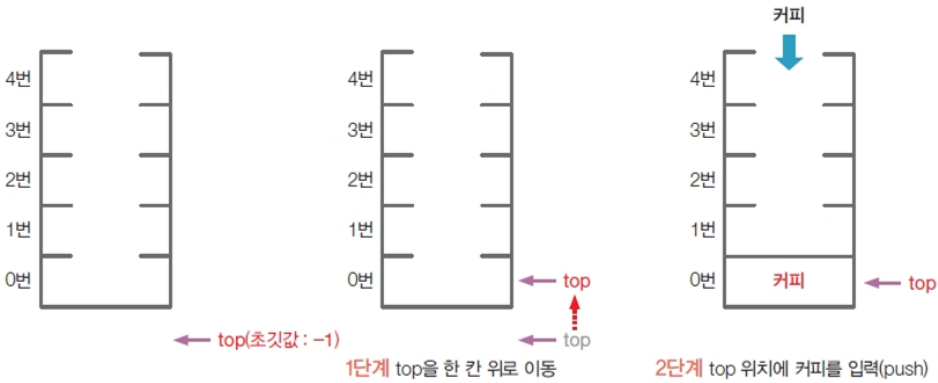
스택(stack)이란 자료구조의 한 형태로, 후입선출(LIFO, Last In First Out) 방식으로 데이터를 처리하는 구조이다. 스택에서 가장 나중에 삽입된 데이터가 가장 먼저 삭제된다. 이를 비유하자면 접시를 쌓아 올릴 때 가장 위에 놓인 접시를 먼저 꺼내는 것과 같다.



스택의 기본 구조

- push: 스택에 데이터를 삽입
- pop: 스택의 데이터를 추출
- top: 스택의 가장 위에 있는 데이터

먼저, 스택에 데이터를 삽입(push)하는 과정이다. 초기 스택은 비어 있고, top의 값은 -1로, 스택이 비어 있음을 나타낸다. 스택의 각 위치는 0번부터 4번까지 번호가 매겨져 있다.



데이터를 삽입하기 위해 top의 값을 1 증가시켜 top의 값은 0이 된다. 이는 새로운 데이터가 삽입될 위치를 나타낸다. 현재 top 값인 0번 위치에 데이터를 삽입(push)한다. 위 그림에서는 “커피”라는 데이터를 0번 위치에 삽입한다. 삽입 후, top 값은 여전히 0을 가리킨다.

다음으로, 스택에서 데이터를 추출(pop)하는 과정이다. pop은 현재 top 값이 가리키는 위치의 데이터를 추출한다. 아래 그림의 예시를 보시면 현재 top의 값은 1번을 가리키고 있으며, 해당 위치에 “녹차”라는 데이터가 있다. pop 연산을 수행하여 top 위치의 데이터를 추출한다. 따라서, “녹차”라는 데이터가 추출된다.

데이터를 추출한 후, top의 값을 1 감소시킨다. 그러면 top의 값은 0이 되고, 다음 pop 연산 시 추출될 데이터의 위치를 가리킨다.

지금까지 우리는 스택의 개념에 대해 알아보았다. 이제 실습을 통해 스택의 기본 연산인 push, pop에 대해 직접 구현하고 사용해 봅시다.

크기가 5칸인 스택의 생성과 데이터 3개 입력

```
1  stack = [None, None, None, None, None]
2  top = -1
3
4  top += 1
5  stack[top] = "커피"
6  top += 1
7  stack[top] = "녹차"
8  top += 1
9  stack[top] = "꿀물"
10
11 print("----- 스택 상태 -----")
12 for i in range(len(stack)-1, -1, -1):
13     print(stack[i])
```

출력 결과

```
----- 스택 상태 -----
None
None
꿀물
녹차
커피
```

✓ **1~2행** 크기가 5인 빈 스택을 생성하고, 스택의 초기 값을 -1로 설정한다.

✓ **4~9행** 스택에 데이터를 삽입하는 과정이다. 각 데이터 삽입 시마다 top 값을 1씩 증가시키고, 해당 위치에 데이터를 저장한다.

✓ **12~13행** 스택의 현재 상태를 출력한다. 스택의 맨 위에서부터 아래로 내려가며 각 요소를 출력한다.

스택에서 데이터 3개 추출

```
1  stack = ["커피", "녹차", "꿀물", None, None]
2  top = 2
3
4  print("----- 스택 상태 -----")
5  for i in range(len(stack)-1, -1, -1):
6      print(stack[i])
7
8  print("---------------------")
9
10 data = stack[top]
11 stack[top] = None
12 top -= 1
13 print("pop -->", data)
14
15 data = stack[top]
16 stack[top] = None
17 top -= 1
18 print("pop -->", data)
19
20 data = stack[top]
21 stack[top] = None
22 top -= 1
23 print("pop -->", data)
24
25 print("---------------------")
26 print("----- 스택 상태 -----")
27 for i in range(len(stack)-1, -1, -1):
28     print(stack[i])
```

출력 결과

```
----- 스택 상태 -----
None
None
꿀물
녹차
커피
--------------------
pop --> 꿀물
pop --> 녹차
pop --> 커피
--------------------
----- 스택 상태 -----
None
None
None
None
None
```

✓ **1~2행** 기존에 데이터가 입력된 스택을 생성하고, 스택의 현재 top 값을 2로 설정한다. 이는 마지막 데이터가 2번 인덱스에 있음을 나타낸다.

✓ **5~6행** 스택의 맨 위에서부터 아래로 내려가며 스택의 현재 상태를 출력한다.

✓ **10~23행** 스택에서 데이터를 추출하는 과정이다. 현재 top 위치의 데이터를 추출하여 변수 data에 저장한 뒤, 현재 top 위치를 None으로 설정한다. top의 값을 1 감소시키고 "pop -->"와 함께 추출된 데이터를 출력한다.

✓ **27~28행** 스택의 맨 위에서부터 아래로 내려가며 스택의 현재 상태를 출력한다.

지금까지 스택의 기본적인 동작 과정을 실습으로 구현해 보았다. 이제 전체 코드를 통해 push(), pop(), 스택이 꽉 찼는지 확인하는 isStackFull(), 스택이 비었는지 확인하는 isStackEmpty(), 스택의 top에 있는 값을 확인하는 peek() 함수를 배워보도록 하겠다.

스택의 전체 코드

```
1  def isStackFull():  # 스택이 꽉 찼는지 확인하는 함수
2      global SIZE, stack, top
3      if (top >= SIZE - 1):
4          return True
5      else:
6          return False
7
8  def isStackEmpty():  # 스택이 비었는지 확인하는 함수
9      global SIZE, stack, top
10     if (top == -1):
11         return True
12     else:
13         return False
14
15 def push(data):  # 스택에 데이터를 삽입하는 함수
16     global SIZE, stack, top
17     if isStackFull():
18         print("스택이 꽉 찼습니다.")
19         return
20     top += 1
21     stack[top] = data
22
23 def pop():  # 스택에서 데이터를 추출하는 함수
24     global SIZE, stack, top
25     if isStackEmpty():
26         print("스택이 비었습니다.")
27         return None
28     data = stack[top]
29     stack[top] = None
30     top -= 1
31     return data
32
```

```
33  def peek():  # 스택에서 top 위치의 데이터를 확인하는 함수
34      global SIZE, stack, top
35      if isStackEmpty():
36          print("스택이 비었습니다.")
37          return None
38      return stack[top]
39
40  # 전역 변수 선언 부분
41  SIZE = int(input("스택 크기를 입력하세요 ==> "))
42  stack = [None for _ in range(SIZE)]
43  top = -1
44
45  # 메인 코드 부분
46  if __name__ == "__main__":
47      select = input("삽입(I)/추출(E)/확인(V)/종료(X) 중 하나를 선택 ==> ")
48
49      while (select != 'X' and select != 'x'):
50          if select == 'I' or select == 'i':
51              data = input("입력할 데이터 ==> ")
52              push(data)
53              print("스택 상태 : ", stack)
54          elif select == 'E' or select == 'e':
55              data = pop()
56              print("추출된 데이터 ==> ", data)
57              print("스택 상태 : ", stack)
58          elif select == 'V' or select == 'v':
59              data = peek()
60              print("확인된 데이터 ==> ", data)
61              print("스택 상태 : ", stack)
62          else:
63              print("입력이 잘못됨")
64
65          select = input("삽입(I)/추출(E)/확인(V)/종료(X) 중 하나를 선택 ==> ")
66
67      print("프로그램 종료!")
```

✓ **1~6행** 스택이 꽉 찼는지 확인하는 함수이다. 전역 변수 SIZE, stack, TOP을 사용하며 TOP이 SIZE-1보다 크거나 같으면 스택이 꽉 찼다고 판단하고 True를 반환한다. 그렇지 않다면 False를 반환한다.

✓ **8~13행** 스택이 비었는지 확인하는 함수이다. 전역 변수 SIZE, stack, TOP을 사용하며 top이 -1이면 스택이 비었다고 판단하고 True를 반환한다. 그렇지 않다면 False를 반환한다.

✓ **15~21행** 데이터를 삽입하는 함수이다. 전역 변수 SIZE, stack, TOP을 사용하며 isStackFull() 함수를 호출하여 스택이 꽉 찼는지 확인하고 꽉 찼다면 메시지를 출력하고 종료한다. 그렇지 않다면 top을 1 증가시키고 top의 위치에 데이터를 삽입한다.

✓ **23~31행** 데이터를 추출하는 함수이다. 전역 변수 SIZE, stack, TOP을 사용하며 isStackEmpty() 함수를 호출하여 스택이 비었는지 확인하고 비었다면 메시지를 출력하고 None을 반환한다. 그렇지 않다면 top 위치의 데이터를 변수 data에 저장하고, top의 데이터를 None으로 설정하고 top의 값을 1 감소시킨 후 데이터를 반환한다.

✓ **33~38행** 스택에서 top 위치의 데이터를 확인하는 함수이다. 전역 변수 SIZE, stack, TOP을 사용하며 isStackEmpty() 함수를 호출하여 스택이 비었는지 확인하고, 비었다면 메시지를 출력 후 None을 반환한다. 그렇지 않다면 top 위치의 데이터를 반환한다.

✓ **45~67행** 메인 코드 부분이다. while문을 사용하여 사용자가 'X' 또는 'x'를 입력할 때까지 반복한다. 사용자가 'I' 또는 'i'를 입력한 경우 데이터를 입력받아 스택에 삽입하고 스택의 상태를 출력한다. 사용자가 'E' 또는 'e'를 입력한 경우 스택에서 추출하여 추출한 값을 출력하고 스택의 상태를 출력한다. 사용자가 'V' 또는 'v'를 입력한 경우 스택의 맨 위 데이터를 출력하고 스택의 상태를 출력한다.

02. 큐?

큐 구조란?

큐(queue)구조란 차량이 한 대 간격으로 줄을 지어 통행이 가능한 터널처럼 먼저 들어간 차량이 먼저 나오는 구조를 말한다. (선입선출)

위 그림에 있는 차량은 A, B, C 순으로 들어갔으며 차량이 줄을 지어 통행이 가능하기 때문에 나오는 순서도 A, B, C 순으로 나오게 된다.

스택의 실생활 사례

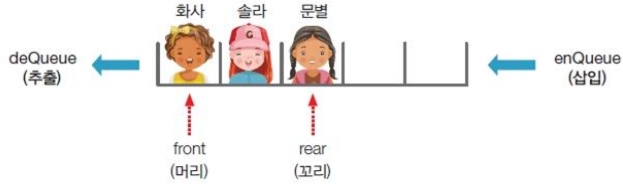
- 은행의 ATM기 줄 서기
- 프린터의 출력 처리
- 콜센터 고객 대기 시간
- 너비 우선 탐색(BFS)

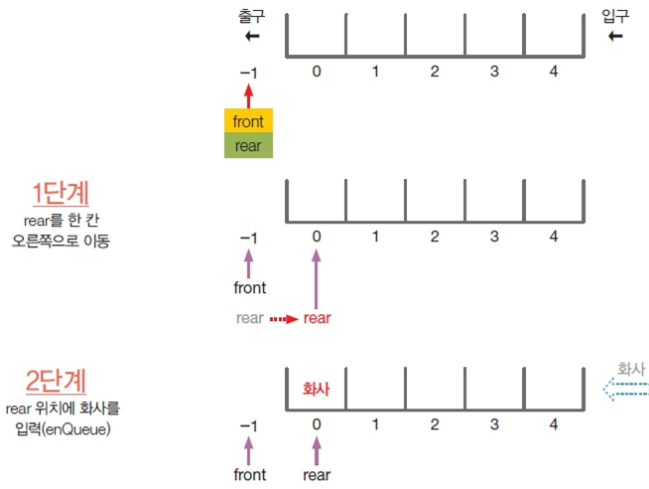
큐?

큐(stack)이란 자료구조의 한 형태로, 선입선출(FIFO, First In First Out) 방식으로 데이터를 처리하는 구조이다. 큐에서 가장 먼저 삽입된 데이터가 가장 먼저 삭제된다. 이를 비유하자면 은행의 대기줄을 생각할 수 있다. 고객들이 줄을 서서 기다리다가, 먼저 줄을 선 고객이 먼저 서비스를 받는 것과 동일한 원리이다.

스택의 기본 구조

- enQueue: 큐에 데이터를 삽입
- deQueue: 큐의 데이터를 추출
- front: 저장된 데이터 중 첫 번째 데이터
- rear: 저장된 데이터 중 마지막 데이터





먼저, 위 그림은 큐에 데이터를 삽입(enQueue)하는 과정이다. 초기 큐는 비어 있고, front와 rear는 -1을 가리키고 있다. 데이터 삽입을 위해 rear를 1 증가시키면 rear는 0번째 인덱스를 가리키게 된다. rear의 위치에 데이터를 삽입한다. 위의 예시에서는 "화사"를 삽입한다.

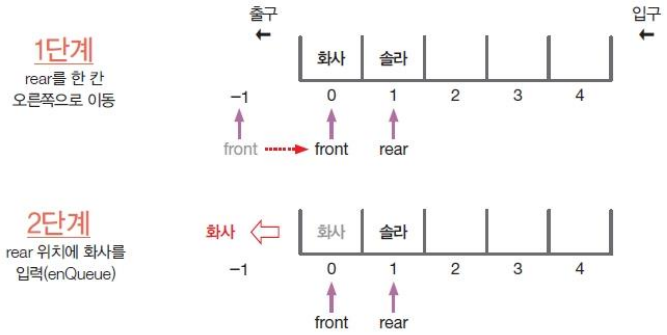


그림 7-6 큐에서 데이터를 추출하는 과정

다음으로, 위 그림은 큐에서 데이터를 추출(dequeuer)하는 과정이다. 초기 큐에는 "화사"와 "솔라"가 있으며, front는 -1번 인덱스, rear는 1번 인덱스를 가리키고 있다. 데이터 추출을 위해 front를 1 증가시키면 front는 0번 인덱스를 가리키게 된다. front가 가리키는 위치의 데이터를 추출한다. 위 예시에서는 "화사"를 추출하게 된다.

지금까지 우리는 큐의 개념에 대해 알아보았다. 이제 실습을 통해 큐의 기본 연산인 enQueue, deQueue에 대해 직접 구현하고 사용해 봅시다.

크기가 5칸인 큐의 생성과 데이터 3개 입력

```
1  queue = [None, None, None, None, None]
2  front = rear = -1
3
4  rear += 1
5  queue[rear] = "화사"
6
7  rear += 1
8  queue[rear] = "솔라"
9
10 rear += 1
11 queue[rear] = "문별"
12
13 print("----- 큐 상태 -----")
14 print('[출구] <- ', end = ' ')
15 for i in range(0, len(queue), 1):
16     print(queue[i], end = ' ')
17 print('<- [입구]')
```

출력 결과

```
----- 큐 상태 ----- [출구] <- 화사 솔라 문별 None None <- [입구]
```

- ✓ **1~2행** 크기가 5인 빈 큐를 생성하고, front와 rear을 -1로 설정한다.
- ✓ **4~5행** rear를 1 증가시켜 0번 인덱스에 “화사”를 삽입한다.
- ✓ **7~8행** rear를 1 증가시켜 1번 인덱스에 “솔라”를 삽입한다.
- ✓ **10~11행** rear를 1 증가시켜 2번 인덱스에 “문별”을 삽입한다.
- ✓ **13~17행** 데이터를 차례대로 삽입한 뒤 큐의 상태를 출력한다.

큐에서 데이터 3개 추출

```
1  queue = ["화사", "솔라", "문별", None, None]
2  front = -1
3  rear = 2
4
5  print("----- 큐 상태 -----")
6  print('[출구] <- ', end=' ')
7  for i in range(0, len(queue), 1):
8      print(queue[i], end=' ')
9  print('<- [입구]')
10 print("--------------------")
11
12 front += 1
13 data = queue[front]
14 queue[front] = None
15 print('deQueue --> ', data)
16
17 front += 1
18 data = queue[front]
19 queue[front] = None
20 print('deQueue --> ', data)
21
22 front += 1
23 data = queue[front]
24 queue[front] = None
25 print('deQueue --> ', data)
26
27 print("--------------------")
28 print("----- 큐 상태 -----")
29 print('[출구] <- ', end=' ')
30 for i in range(0, len(queue), 1):
31     print(queue[i], end=' ')
32 print('<- [입구]')
```

출력 결과

```
----- 큐 상태 -----
[출구] <- 화사 솔라 문별 None None <- [입구]
--------------------
deQueue --> 화사
deQueue --> 솔라
deQueue --> 문별
--------------------
----- 큐 상태 -----
[출구] <- None None None None None <- [입구]
```

- ✓ **1~3행** 큐를 초기화하고, front와 rear의 값을 설정한다.
- ✓ **5~10행** 현재 큐의 상태를 출력한다.
- ✓ **12~15행** front의 값을 1 증가시킨다. 큐의 첫 번째 원소를 가리키게 된다. front 위치의 데이터를 data 변수에 저장한다. 현재 front는 0이므로 queue[0]의 값인 "화사"가 data에 저장된다. 그리고 추출한 데이터의 위치인 queue[0]을 None으로 설정하고, 추출한 데이터를 출력한다.
- ✓ **17~20행** front의 값을 1 증가시킨다. 큐의 두 번째 원소를 가리키게 된다. front 위치의 데이터를 data 변수에 저장한다. 현재 front는 1이므로 queue[1]의 값인 "솔라"가 data에 저장된다. 그리고 추출한 데이터의 위치인 queue[1]을 None으로 설정하고, 추출한 데이터를 출력한다.
- ✓ **22~25행** front의 값을 1 증가시킨다. 큐의 세 번째 원소를 가리키게 된다. front 위치의 데이터를 data 변수에 저장한다. 현재 front는 2이므로 queue[2]의 값인 "문별"가 data에 저장된다. 그리고 추출한 데이터의 위치인 queue[2]을 None으로 설정하고, 추출한 데이터를 출력한다.
- ✓ **27~32행** 최종적으로 큐의 상태를 출력한다.

지금까지 큐의 기본적인 동작 과정을 실습으로 구현해 보았다. 이제 전체 코드를 통해 euQueue(), deQueue(), 큐가 꽉 찼는지 확인하는 isQueueFull(), 큐가 비었는지 확인하는 isQueueEmpty(), 추출될 데이터를 확인하는 peek() 함수를 배워보도록 하자.

큐의 전체 코드

```
1  def isQueueFull():
2      global SIZE, queue, front, rear
3      if (rear != SIZE-1):
4          return False
5      elif (rear == SIZE-1) and (front == -1):
6          return True
7      else:
8          for i in range(front+1, SIZE):
9              queue[i-1] = queue[i]
10             queue[i] = None
11         front -= 1
12         rear -= 1
13         return False
14
15 def isQueueEmpty():
16     global SIZE, queue, front, rear
17     if (front == rear):
18         return True
19     else:
20         return False
21
22 def enQueue(data):
23     global SIZE, queue, front, rear
24     if (isQueueFull()):
25         print("큐가 꽉 찼습니다.")
26         return
27     rear += 1
28     queue[rear] = data
29
```

```
30  def deQueue():
31      global SIZE, queue, front, rear
32      if (isQueueEmpty()):
33          print("큐가 비었습니다.")
34          return None
35      front += 1
36      data = queue[front]
37      queue[front] = None
38      return data
39
40  def peek():
41      global SIZE, queue, front, rear
42      if (isQueueEmpty()):
43          print("큐가 비었습니다.")
44          return None
45      return queue[front+1]
46
47  # 전역 변수 선언 부분
48  SIZE = int(input("큐 크기를 입력하세요 ==> "))
49  queue = [None for _ in range(SIZE)]
50  front = rear = -1
51
52  # 메인 코드 부분
53  if __name__ == "__main__":
54      select = input("삽입(I)/추출(E)/확인(V)/종료(X) 중 하나를 선택 ==> ")
55
56      while (select != 'X' and select != 'x'):
57          if select == 'I' or select == 'i':
58              data = input("입력할 데이터 ==> ")
59              enQueue(data)
60              print("큐 상태 : ", queue)
61          elif select == 'E' or select == 'e':
62              data = deQueue()
63              print("추출된 데이터 ==> ", data)
64              print("큐 상태 : ", queue)
65          elif select == 'V' or select == 'v':
66              data = peek()
67              print("확인된 데이터 ==> ", data)
68              print("큐 상태 : ", queue)
69          else:
70              print("입력이 잘못됨")
71          select = input("삽입(I)/추출(E)/확인(V)/종료(X) 중 하나를 선택 ==> ")
72      print("프로그램 종료!")
```

✓ **1~13행** 큐가 꽉 찼는지 확인하는 함수이다. 전역 변수 SIZE, queue, front, rear를 사용한다. rear가 SIZE-1이 아니면 큐가 꽉 차지 않았으므로 False를 반환한다. rear가 SIZE-1이고, front가 -1이면 큐가 꽉 찼음을 나타내기에 True를 반환한다. 그 외의 경우에는 큐의 데이터를 앞으로 한 칸씩 이동시키고 front, rear를 1씩 감소시킨 후 False를 반환한다.

✓ **15~20행** 큐가 비었는지 확인하는 함수이다. 전역 변수 SIZE, queue, front, rear를 사용한다. front와 rear가 같으면 큐가 비었음을 나타내며 True를 반환한다. 그렇지 않다면 False를 반환한다.

✓ **22~28행** 큐의 데이터를 삽입하는 함수이다. 전역 변수 SIZE, queue, front, rear를 사용한다. isQueueFull 함수를 호출하여 큐가 꽉 찼는지 확인하고, 꽉 찼으면 메시지를 출력하고 종료한다. 그렇지 않다면 rear를 1 증가시키고 해당 위치에 데이터를 삽입한다.

✓ **30~38행** 큐의 데이터를 추출하는 함수이다. 전역 변수 SIZE, queue, front, rear를 사용한다. isQueueEmpty() 함수를 호출하여 큐가 비었는지 확인하고 비었다면 메시지를 출력하고 None을 반환한다. 그렇지 않다면 front를 1 증가시키고 해당 위치의 데이터를 추출하여 반환한다.

✓ **40~45행** 큐에서 추출될 데이터를 확인하는 함수이다. 전역 변수 SIZE, queue, front, rear를 사용한다. isQueueEmpty 함수를 호출하여 큐가 비었는지 확인하고, 비었다면 메시지를 출력하고 None을 반환한다. 그렇지 않다면 다음에 추출될 데이터 front+1 위치의 데이터를 반환한다.

✓ **47~50행** 큐의 크기를 사용자로부터 입력받고, 그 크기만큼의 리스트를 None으로 초기화하여 큐를 생성한다. front와 rear를 -1로 초기화한다.

✓ **52~74행** 메인 코드 부분이다. while문을 사용하여 사용자가 'X' 또는 'x'를 입력할 때까지 반복한다. 사용자가 'I' 또는 'i'를 입력한 경우 데이터를 입력받아 큐에 삽입하고 큐의 상태를 출력한다. 사용자가 'E' 또는 'e'를 입력한 경우 큐에서 데이터를 추출하여 추출한 값을 출력하고 큐의 상태를 출력한다. 사용자가 'V' 또는 'v'를 입력한 경우 큐에서 다음에 추출될 데이터를 출력하고, 현재 큐의 상태를 출력한다.

✓ **69~70행** 사용자가 잘못된 입력을 한 경우 오류 메시지를 출력한다.

✓ **76행** 사용자로부터 'X' 또는 'x' 값을 받기 전까지 반복해서 입력을 받는다.

제 5 장

기본 정렬 알고리즘

1) 정렬이란?
2) 선택 정렬
3) 삽입 정렬
4) 버블 정렬

01. 정렬이란?

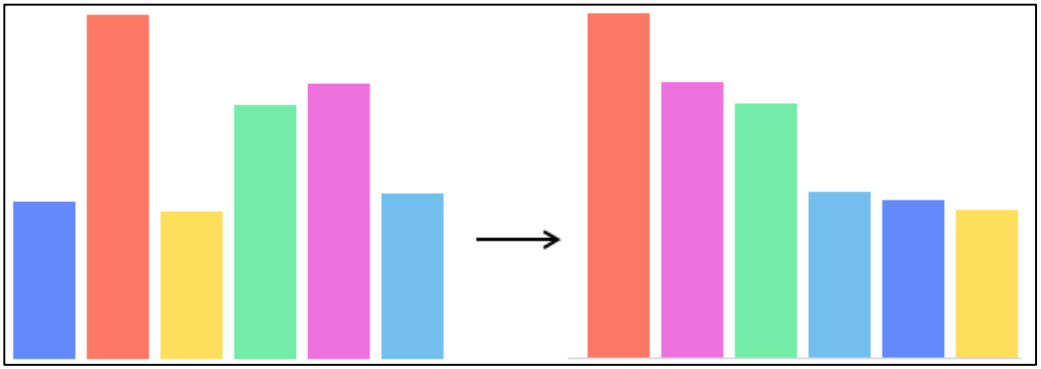
정렬의 정의

정렬(sorting)이란 데이터를 특정한 조건에 따라 일정한 순서가 되도록 다시 배열하는 것을 뜻한다. 순서대로 나열할 때는 작은 것부터 나열하는 방법과 큰 것부터 나열하는 방법이 있다. 이는 숫자, 문자열 또는 다른 데이터 타입이 될 수 있다. 정렬을 사용하면 데이터 탐색이 용이하고, 데이터셋 비교가 용이하다.

정렬 알고리즘의 중요성

정렬 알고리즘은 데이터베이스 시스템, 검색 엔진, 머신러닝 알고리즘 등 다양한 분야에서 활용되고 있다.

효율적인 정렬 알고리즘을 사용하면 시간 복잡도(Time Complexity)를 최소화하여 데이터 처리 속도를 크게 향상시킬 수 있다.



기본적으로 파이썬에서 sort() 메소드와 sorted() 내장함수를 지원한다.

파이썬 정렬 함수 사용 예제
리스트명.sort() sorted(리스트명)

sort()함수는 리스트명.sort()형식으로 이용하는 리스트형의 메소드이다.

즉, 이 함수는 입력 값으로 리스트밖에 받지 못한다. 또, 이 함수는 리스트 원본 값을 직접 수정한다. 또, sorted 함수와는 다르게, 정렬을 수행한 리스트를 따로 반환하지 않는다.

sorted()함수는 sorted(리스트명) 형식으로 이용하는 내장 함수이며, 리스트 원본 값을 그대로 유지한다. sort 함수와는 다르게 어떤 입력 값이든 받을 수 있다. 또, 이 함수는 정렬 작업을 수행한 후의 값을 반환한다.

그럼 둘 중 뭘 쓰면 되나요?

- 원본을 유지할 때는 sorted() 함수 사용
- 리스트 말고 다른 자료형을 정렬할 때는 sorted() 함수 사용
- 원본이 필요 없을 때는 시간, 공간 절약을 위해 sort() 함수 사용
- list를 이용할 때는 시간, 공간 절약을 위해 sort() 함수 사용

정렬하는 방법에 대한 정렬 알고리즘은 수십 가지이다. 이해하고 구현하기 쉽지만 속도가 느린 알고리즘, 이해와 구현하기 어렵지만 속도가 빠른 알고리즘, 특수한 상황에서만 효율적인 알고리즘, 메모리를 적게 사용하는 알고리즘 등 다양하다. 상황에 맞는 정렬 알고리즘을 선택하는 것이 중요하다.

정렬 알고리즘 선택 시 고려사항

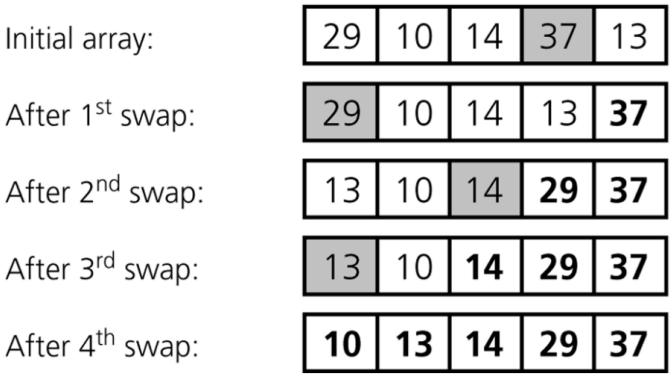
- **시간복잡도**
  - ✓ 알고리즘이 입력 데이터의 크기에 따라 얼마나 빨리 실행되는지
- **메모리 사용량**
  - ✓ 알고리즘이 추가적인 메모리를 얼마나 사용하는지
- **안정성**
  - ✓ 동일한 값의 요소들이 정렬 후에도 원래의 순서를 유지하는지

다음 장에서는 기본적인 정렬 알고리즘에 대해 소개하고, 각 정렬 알고리즘의 자세한 동작 원리와 구현 방법에 대해 살펴보도록 하겠다.

02. 선택 정렬

선택 정렬(Selection Sort)은 단순한 정렬 시리즈 중 하나로 간단하고 직관적인 정렬 알고리즘이다.

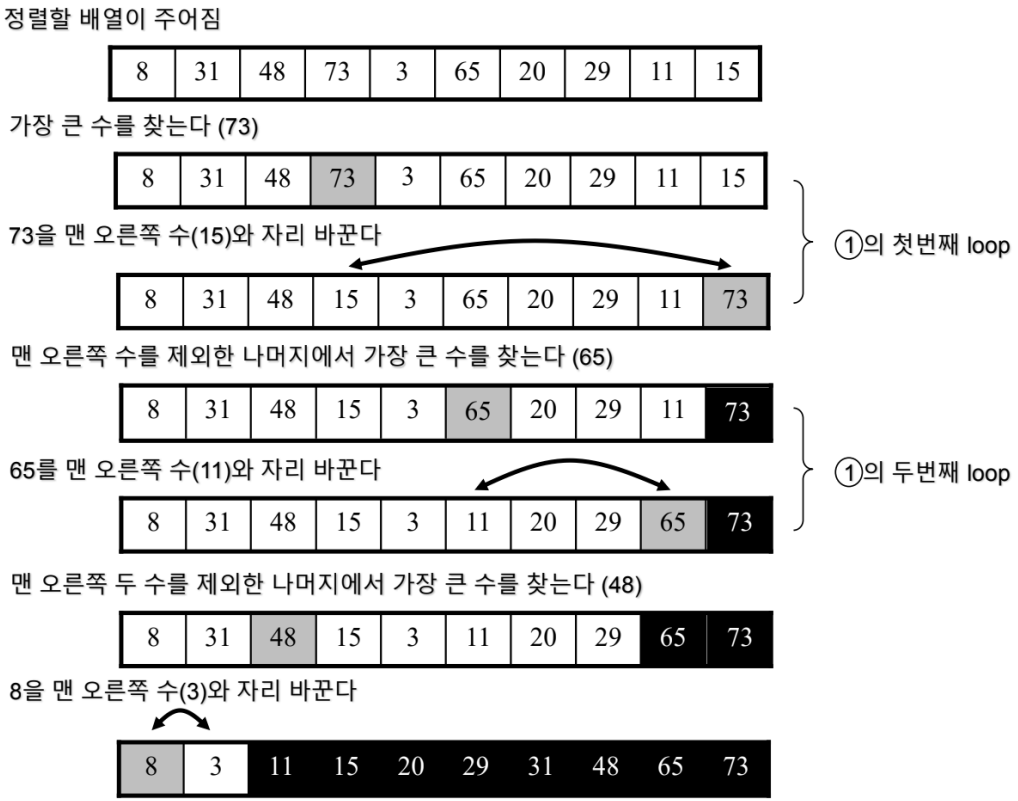
선택 정렬은 특정 숫자(최대 또는 최소)를 선택하는 방식으로 정렬을 진행한다. 또한, 제자리 정렬(In-place sorting) 알고리즘으로 입력 배열 이외에 다른 추가 메모리를 요구하지 않는다.



선택 정렬의 동작 방식

1. 주어진 리스트에서 최대 원소를 선택한다.
2. 최대 원소와 맨 오른쪽 원소를 교환한다. (오름차순)
3. 맨 오른쪽 원소를 제외하여 하나의 원소만 남을 때가지 위의 동작을 반복한다.

선택 정렬 동작과정


정렬할 배열이 주어짐
8 31 48 73 3 65 20 29 11 15
가장 큰 수를 찾는다 (73)
8 31 48 73 3 65 20 29 11 15
73을 맨 오른쪽 수(15)와 자리 바꾼다
8 31 48 15 3 65 20 29 11 73
①의 첫번째 loop
맨 오른쪽 수를 제외한 나머지에서 가장 큰 수를 찾는다 (65)
8 31 48 15 3 65 20 29 11 73
65를 맨 오른쪽 수(11)와 자리 바꾼다
8 31 48 15 3 11 20 29 65 73
①의 두번째 loop
맨 오른쪽 두 수를 제외한 나머지에서 가장 큰 수를 찾는다 (48)
8 31 48 15 3 11 20 29 65 73
8을 맨 오른쪽 수(3)와 자리 바꾼다
8 3 11 15 20 29 31 48 65 73

선택 정렬 파이썬 코드

```
1  def Selection_Sort(ary):
2          for i in range(len(ary)-1,0,-1):
3                  max = i
4                  for j in range(i):
5                          if ary[max] < ary[j]:
6                                  max = j
7                  ary[i], ary[max] = ary[max], ary[i]
8          return ary
```

- ✓ 2행 리스트의 마지막 원소부터 시작하여 처음까지 반복
- ✓ 3행 현재 위치를 최댓값의 위치로 설정
- ✓ 4행 리스트의 처음 위치부터 현재 위치까지 반복
- ✓ 5행 현재 최댓값이 현재 원소보다 작으면
- ✓ 6행 최댓값의 위치를 현재 원소의 위치로 업데이트
- ✓ 7행 최댓값과 현재 위치의 원소를 교환

수행 시간

- ✓ 2행 for 루프는 '배열크기-1'번 반복
- ✓ 6행 가장 큰 수를 찾기 위한 비교횟수: n-1, n-2,...,2, 1
- ✓ 7행 원소 교환은 상수 시간 작업

시간복잡도 (비교횟수 관점)

- ✓ $T(n) = (n-1) + (n-2) + \ ... + 2 + 1 = O(n^2)$

정리

- 선택 정렬은 제자리 정렬 알고리즘의 하나로 입력 배열 이외에 다른 추가 메모리를 요구하지 않는 정렬이다.

- 해당 순서에 원소를 넣을 위치는 이미 정해져 있고, 어떤 원소를 넣을지 선택하는 알고리즘이다.

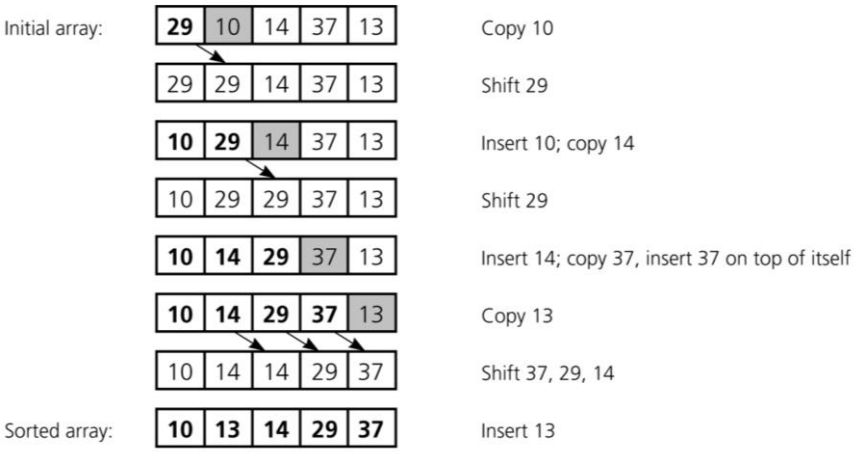
- 주어진 리스트가 거의 정렬되어 있든지, 역으로 정렬되어 있든지, 랜덤하게 되어 있는지 구분하지 않고, 항상 일정한 시간 복잡도를 나타낸다.

- 그러나 시간복잡도가 $O(n^2)$으로 크기 때문에 큰 데이터셋에는 비효율적이다.

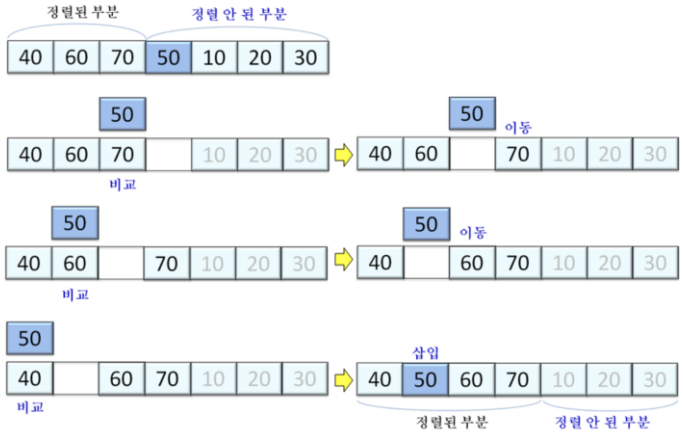
- 추가적인 메모리 사용이 없다는 점은 장점이지만, 안정적이지 않다는 단점이 있다.

03. 삽입 정렬

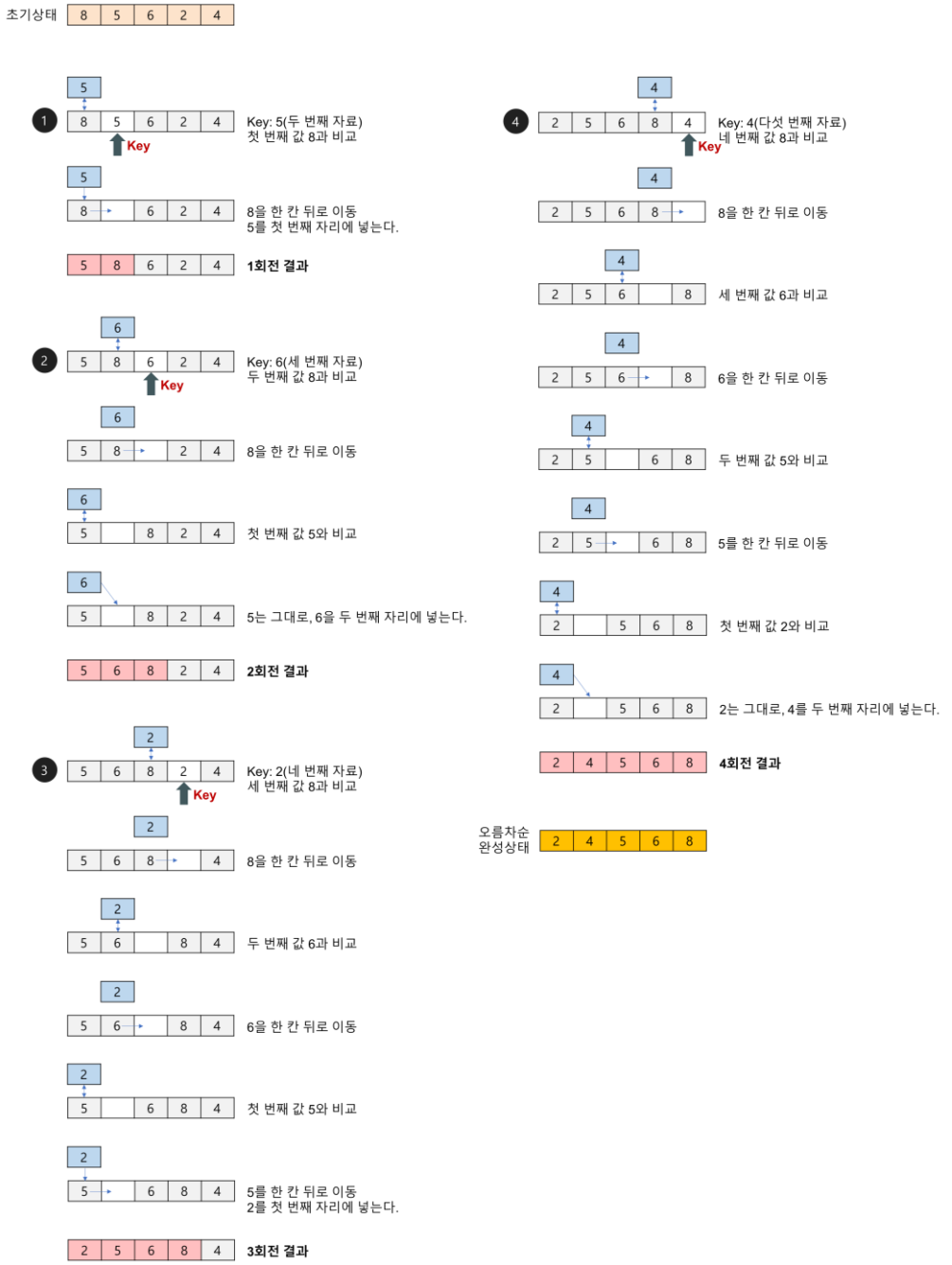
삽입 정렬(Selection Sort)은 단순한 정렬 시리즈 중 하나로 간단하고 직관적인 정렬 알고리즘이다.



삽입 정렬(Insert Sort)은 배열을 정렬된 부분 (앞부분)과 정렬 안 된 부분 (뒷부분)으로 나누고, 정렬 안된 부분의 가장 왼쪽 원소를 정렬된 부분의 적절한 위치에 삽입하여 정렬되도록 하는 과정을 반복한다.



삽입 정렬 동작과정


초기상태
8 5 6 2 4
1
Key: 5(두 번째 자료)
첫 번째 값 8과 비교
Key
8을 한 칸 뒤로 이동
5를 첫 번째 자리에 넣는다.
1회전 결과
2
Key: 6(세 번째 자료)
두 번째 값 8과 비교
8을 한 칸 뒤로 이동
첫 번째 값 5와 비교
5는 그대로, 6을 두 번째 자리에 넣는다.
2회전 결과
3
Key: 2(네 번째 자료)
세 번째 값 8과 비교
8을 한 칸 뒤로 이동
두 번째 값 6과 비교
6을 한 칸 뒤로 이동
첫 번째 값 5와 비교
5를 한 칸 뒤로 이동
2를 첫 번째 자리에 넣는다.
3회전 결과
4
Key: 4(다섯 번째 자료)
네 번째 값 8과 비교
8을 한 칸 뒤로 이동
세 번째 값 6과 비교
6을 한 칸 뒤로 이동
두 번째 값 5와 비교
5를 한 칸 뒤로 이동
첫 번째 값 2와 비교
2는 그대로, 4를 두 번째 자리에 넣는다.
4회전 결과
오름차순
완성상태
2 4 5 6 8

삽입 정렬 파이썬 코드

```
1  def Insert_Sort(ary):
2          for i in range(1,len(ary)):
3                  for j in range(i,0,-1):
4                          if ary[j] < ary[j-1]:
5                                  ary[j], ary[j-1] = ary[j-1], ary[j]
6          return ary
```

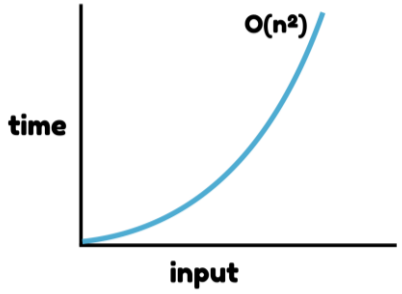
- ✓ 2행 리스트의 두 번째 원소부터 시작하여 끝까지 반복
- ✓ 3행 현재 원소부터 리스트의 시작까지 반복
- ✓ 4행 현재 원소가 이전 원소보다 작으면
- ✓ 5행 현재 원소와 이전 원소를 교환

수행 시간

- ✓ 2행 for 루프는 '배열크기-1'번 반복
- ✓ 3~5행 삽입은 최악의 경우 i-1회 비교

시간복잡도 (비교횟수 관점)

- ✓ $T(n) = (n-1) + (n-2) + \ldots + 2 + 1 = O(n^2)$



정리

- 삽입 정렬은 정렬된 부분 배열을 유지하며, 한 칸씩 늘려가며 정렬한다.
- 정렬 안 된 부분의 숫자 하나가 정렬된 부분에 삽입됨으로써, 정렬된 부분의 원소 수가 1개 늘어나고, 동시에 정렬이 안 된 부분의 원소 수는 1개 줄어든다.
- 삽입 정렬은 입력 상태에 따라 수행시간이 달라질 수 있다.
- 거의 정렬된 입력에 대해서 다른 정렬 알고리즘보다 빠르며, 입력 자료가 역순일 경우에는 최악의 시간복잡도를 가진다.
- 비교적 많은 원소들의 이동을 포함하기 때문에 원소의 수가 많고 특히 원소의 크기가 클 경우에 적합하지 않는다.

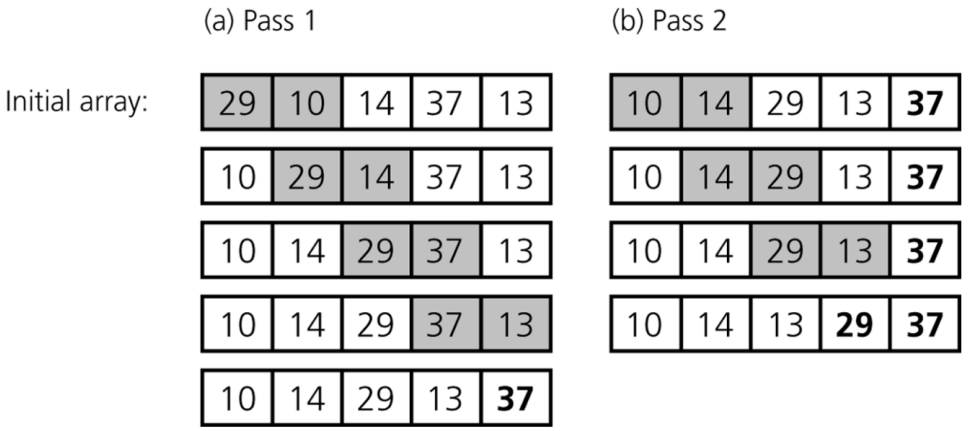
03. 버블 정렬

버블 정렬(Bubble Sort)은 단순한 정렬 시리즈 중 하나로 간단하고 직관적인 정렬 알고리즘이다.

버블 정렬은 이웃하는 숫자를 비교하여 특정한 수(최대 또는 최소)를 한 쪽(앞 또는 뒤)으로 이동시키는 과정을 반복하여 정렬하는 알고리즘이다.

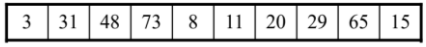
특정 조건을 만족하는 수를 한 쪽으로 보내는 것은 선택 정렬과 동일하고, 특정 조건을 만족하는 수를 찾는 방법은 상이하다.

정렬되는 모습이 마치 거품이 올라오는 모습과 유사하여 버블 정렬이라 불린다.

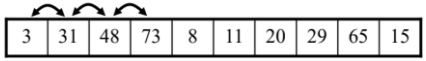


버블 정렬 동작과정

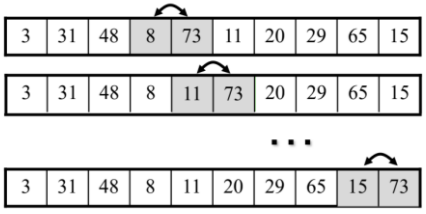
정렬할 배열이 주어진다



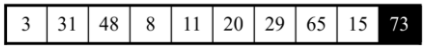
왼쪽부터 시작해 이웃한 쌍들을 비교해 나간다



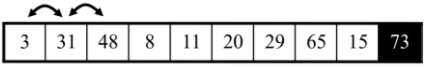
순서대로 되어 있지 않으면 자리 바꾼다



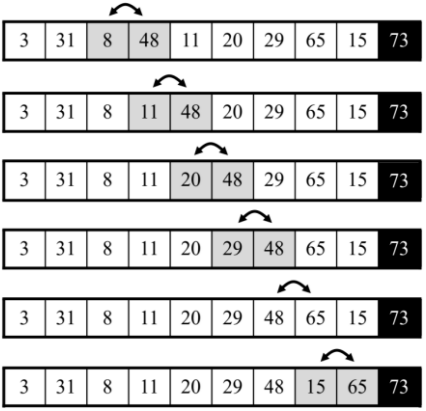
맨 오른쪽 수(73)를 대상에서 제외한다



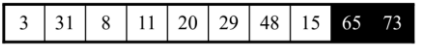
왼쪽부터 시작해 이웃한 쌍들을 비교해간다



순서대로 되어 있지 않은 경우에는 자리 바꾼다



맨 오른쪽 수(65)를 대상에서 제외한다



앞의 작업을 반복하면서 계속 제외해 나간다

두개짜리 배열의 처리를 끝으로 정렬이 완료된다

버블 정렬 파이썬 코드

```
1  def Bubble_Sort(ary):
2          for i in range(len(ary)-1,-1,-1):
3                  for j in range(i):
4                          if ary[j] > ary[j+1]:
5                                  ary[j], ary[j+1] = ary[j+1], ary[j]
6          return ary
```

- ✓ 2행 리스트의 마지막 원소부터 시작하여 처음까지 반복
- ✓ 3행 리스트의 처음 위치부터 현재 위치까지 반복
- ✓ 4행 현재 원소가 다음 원소보다 크면
- ✓ 5행 현재 원소와 다음 원소 교환

수행 시간

- ✓ 2행 for 루프는 '배열크기-1'번 반복
- ✓ 3행 for 루프는 각각 n-1, n-2, ..., 2, 1번 반복
- ✓ 5행 원소 교환은 상수 시간 작업

시간복잡도 (비교횟수 관점)

- ✓ $T(n) = (n-1) + (n-2) + \ ... + 2 + 1 = O(n^2)$

정리

- 버블 정렬은 구현이 매우 간단한다.
- 이웃하는 숫자를 비교하여 특정한 수(최대 또는 최소)를 한 쪽(앞 또는 뒤)으로 이동시킨다.
- 버블 정렬은 교육용 정렬 방법으로 알려져 있고, 원소 교환이 너무 많아 비효율적이기에 실용성이 없는 알고리즘이다.
- 특히 특정 요소가 최종 정렬 위치에 이미 있던 경우에도 교환되는 일이 발생하고 자료의 교환 작업(SWAP)이 자료의 이동 작업(MOVE)보다 더 복잡하다.

제 6 장

고급 정렬 알고리즘

1) 재귀호출
2) 합병 정렬
3) 퀵 정렬

01. 재귀호출

위 인형은 러시아의 전통 인형 마트료시카이다.

마트료시카는 큰 인형을 열면 그보다 작은 인형이 하나 나오고 다시 그 인형을 열면 또 작은 인형이 나오고를 반복하며 점점 크기가 작아진다.

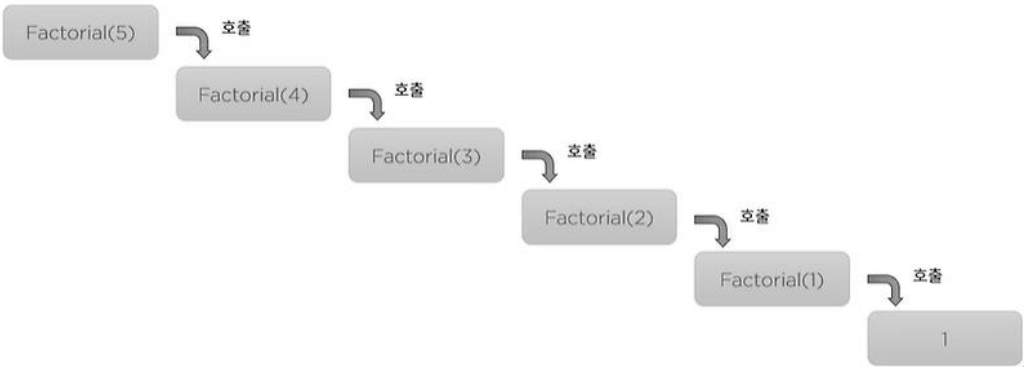
재귀함수도 이와 마찬가지로 주어진 조건을 충족할 때까지 함수 자신의 크기를 줄여가다 조건을 충족하면 결과값을 반환하는 함수이다.

재귀호출(Recursion)은 함수 내부에서 함수가 자기 자신을 또다시 호출(직접재귀함수) 하는 행위를 의미한다.

이러한 재귀 호출은 자기가 자신을 계속해서 호출하므로, 끝없이 반복되게 된다. 파이썬에서는 재귀호출이 너무 많아지면 반복을 자동 종료한다.

일반적인 프로그램에서는 이렇게 무한 반복하는 경우는 거의 없으며, 무한 반복을 마치고 다시 되돌아가는 조건을 함께 사용한다.

재귀 호출은 처음 접하면 상당히 혼란스럽지만 자료구조와 알고리즘을 학습할 때 매우 유용한 방법이므로 꼭 알아 두어야 한다.



재귀 호출의 작동

재귀 호출 함수 기본

```
1  def openBox():
2          print("종이 상자를 엽니다")
3          openBox()
4
5  openBox()
```

출력 결과

```
종이 상자를 엽니다
종이 상자를 엽니다
''' 중략 '''
---------------------------------------------------------------------------
''' 중략 '''
<ipython-input-3-ef9a8252a507> in openBox()
      1 def openBox():
      2     print("종이 상자를 엽니다")
----> 3     openBox()
      4
      5 openBox()

RecursionError: maximum recursion depth exceeded while calling a Python object
```

- ✓ **1행** 함수를 선언한다.
- ✓ **2~3행** 2행에서 메시지를 출력하고 3행에서 다시 자신을 호출한다.
- ✓ **5행** 함수를 처음 호출한다.

이렇듯 재귀 호출에서 종료조건을 넣지 않아 파이썬에서 자동으로 반복을 종료한 것을 확인할 수 있다.

재귀 호출의 작동

재귀 호출 함수 기본(반환 조건 추가)

```
1  def openBox():
2          global count
3          print("종이 상자를 엽니다.")
4          count -= 1
5          if count == 0:
6                  print("** 반지를 넣고 반환합니다. **")
7                  return
8          openBox()
9          print("종이 상자를 닫습니다.")
10
11 count = 10
12 openBox()
```

출력 결과

```
종이 상자를 엽니다.
종이 상자를 엽니다.
'''중략'''
종이 상자를 엽니다.
** 반지를 넣고 반환합니다.
** 종이 상자를 닫습니다
종이 상자를 닫습니다.
'''중략'''
종이 상자를 닫습니다.
종이 상자를 닫습니다.
```

✓ **4, 11행** count를 10회로 설정하는 것은 종이 상자가 10겹이라는 의미이다. 4행에서 종이 상자를 하나씩 꺼내는 작업을 한다.

✓ **5~7행** 5행에서 종이 상자를 모두 꺼냈으면 6행에서 반지를 넣고 7행에서 return문으로 지금까지 호출한 반대로 돌아간다. 즉, 9행으로 돌아가 메시지를 출력한다.

재귀 호출의 연습

팩토리얼을 재귀 호출로 구현

```
1  def factorial(num):
2        if num <= 1:
3              print('1 반환')
4              return 1
5        print("%d * %d! 호출" % (num, num-1))
6        retVal = factorial(num-1)
7
8        print("%d * %d!(=%d) 반환" % (num,num-1,retVal))
9        return num * retVal
10
11 print('5! = ', factorial(5))
```

출력 결과

```
5 * 4! 호출
4 * 3! 호출
3 * 2! 호출
2 * 1! 호출
1 반환
2 * 1!(=1) 반환
3 * 2!(=2) 반환
4 * 3!(=6) 반환
5 * 4!(=24) 반환
5! = 120
```

- ✓ **2~4행** 매개변수 num이 1 이하이면 1을 반환한다.
- ✓ **5행** 매개변수 num이 1 이상이면 "숫자*숫자-1! 호출" 글자를 출력한다.
- ✓ **6행** num-1을 다시 factorial()로 재귀 호출한다. 그리고 retVal에 넣는다.
- ✓ **8행** 재귀 호출이 끝나고 반환되면 "숫자*(숫자-1)! (=계산값) 반환" 글자를 출력한다.
- ✓ **9행** num*(num-1)!의 값을 반환한다.

재귀 호출의 연습

피보나치 수를 재귀 호출로 구현

```
1  def fibo(n):
2          if n == 0:
3                  return 0
4          elif n == 1:
5                  return 1
6          else:
7                  return fibo(n-1) + fibo(n-2)
8
9  print('피보나치 수 --> 0 1', end = ' ')
10 for i in range(2, 20):
11 print(fibo(i), end = ' ')
```

출력 결과

```
피보나치 수 --> 0 1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987 1597 2584 4181
```

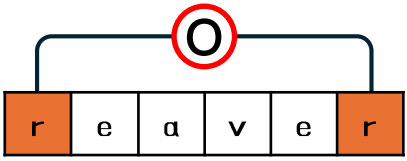
- ✓ **1~7행** 피보나치 수를 반환하는 재귀 함수이다. 매개변수로 피보나치 수의 위치를 전달받는다. 예로 7을 전달받으면 7번째(0부터 카운트) 위치의 피보나치 수를 계산한다.
- ✓ **2~5행** 0번째는 0, 1번째는 1을 반환한다.
- ✓ **6~7행** 0번째와 1번째 이외에는 앞 두 숫자의 합계를 반환한다. 재귀함수를 사용한다.
- ✓ **9행** 0과 1은 그대로 출력한다.
- ✓ **10~11행** 2번째 위치부터 19번째 위치까지 피보나치 수를 구해서 출력한다.

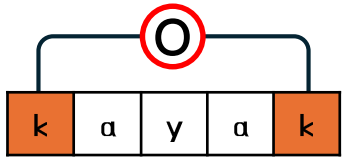
재귀 호출의 응용

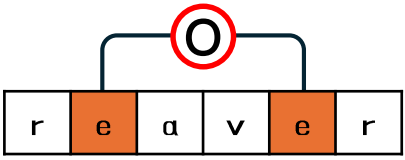
1. 회문 판단하기

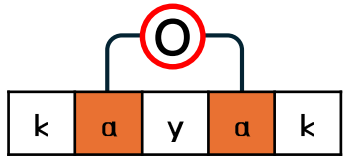
회문(Palindrome)은 앞에서부터 읽든 뒤에서부터 읽든 동일한 단어나 문장을 의미한다. 회문은 첫 글자와 마지막 글자를 비교하여 같으면 회문일 수 있고, 다르면 회문이 아니다. 비교하지 않은 글자가 한 글자 이하이면 회문으로 확정하고, 한 글자 이하가 아니면 첫 글자는 다음 글자로 마지막 글자는 바로 앞 글자로 이동하는 식으로 비교하고 확인하는 과정을 반복한다.

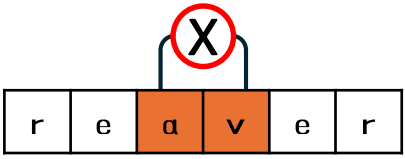
회문이 아닌 경우와 회문인 경우를 예로 살펴보도록 하자.

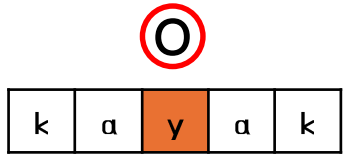












회문 여부를 구별하기

```
1  def palindrome(pStr):
2        if len(pStr) <= 1:
3              return True
4        if pStr[0] != pStr[-1]:
5              return False
6
7        return palindrome(pStr[1:len(pStr)-1])
8
9  ## 전역 변수 선언 부분 ##
10 strAry = ['reaver', "kayak", "Borrow or rob"]
11
12 ## 메인 코드 부분 ##
13 for testStr in strAry:
14       print(testStr, end = '--> ')
15       testStr = testStr.lower().replace(' ','')
16       if palindrome(testStr):
17             print('O')
18       else:
19             print('X')
```

출력 결과

```
reaver--> X
kayak--> O
Borrow or rob--> O
```

✓ 2~3행 문자열의 길이가 1 이하이면 회문으로 판단한다.

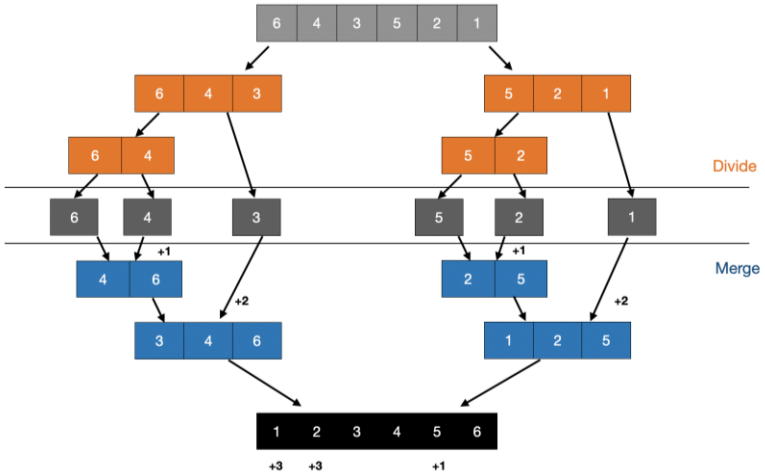
✓ 4~5행 현재 문자열의 맨 앞과 마지막 글자가 다르면 회문이 아닌 것으로 판단한다.

✓ 7행 아직 회문 여부가 결정되지 않았다면, 앞뒤 한 칸씩 잘라 내고 재귀 함수를 호출해서 다시 회문 여부를 확인한다.

02. 합병 정렬

분할정복 알고리즘 (Divide-and-Conquer algorithm)은 그대로 해결할 수 없는 문제를 작은 문제로 분할하여 문제를 해결하는 방법이다.

대표적인 예로는 정렬 알고리즘 중에서 퀵 정렬이나 합병 정렬과 이진 탐색, 선택 문제, 고속 푸리에 변환(FFT) 문제들이 대표적이다.



분할정복 설계

1) Divide
   - ➢ 원래 문제가 분할하여 비슷한 유형의 더 작은 하위 문제로 분할이 가능할 때까지 나눈다.
2) Conquer
   - ➢ 각 하위 문제를 재귀적으로 해결한다. 하위 문제의 규모가 나눌 수 없는 단위가 되면 탈출 조건을 설정하고 해결한다.
3) Combine
   - ➢ Conquer한 문제들을 통합하여 원래 문제의 답을 얻어 해결한다.

Divide를 제대로 나누면 Conquer과정이 자동으로 쉬워진다. 그래서 Divide를 잘 설계하는 것이 중요하다.

분할정복 알고리즘은 재귀 알고리즘이 많이 사용되는데, 이 부분에서 분할정복 알고리즘의 효율성을 저하시킬 수 있다.

분할정복 특징 및 장단점

- 분할된 작은 문제는 입력 크기만 작아지고, 원래 문제와 성격이 동일하다.
- 분할된 문제는 서로 독립적이기에 중복이 제거가 되지 않고, 순환적 분할 및 결과 결합이 가능하다.

장점	단점
✓ Top-down 재귀방식으로 구현하기 때문에 코드가 직관적이다. ✓ 문제를 나누어 해결한다는 특징상 병렬적으로 문제를 해결할 수 있다.	✓ 재귀 함수 호출로 오버헤드가 발생할 수 있다. ✓ 스택에 다량의 데이터가 보관되는 경우 오버플로우가 발생할 수 있다.

합병 정렬(Merge Sort) 알고리즘

존 폰 노이만(John von Neumann)이 1945년에 제안한 방법이며, 일반적인 방법으로 구현했을 때 이 정렬은 안정 정렬에 속하며, 분할 정복 알고리즘의 하나이다.

합병 정렬의 과정은 하나의 리스트를 두 개의 균등한 크기로 분할하고, 분할된 부분 리스트를 정렬한 다음, 두 개의 정렬된 부분 리스트를 합하여 전체가 정렬된 리스트가 되게 한다.

합병 정렬 과정

- 분할(Divide)

  입력 배열을 같은 크기의 2개의 부분 배열로 분할

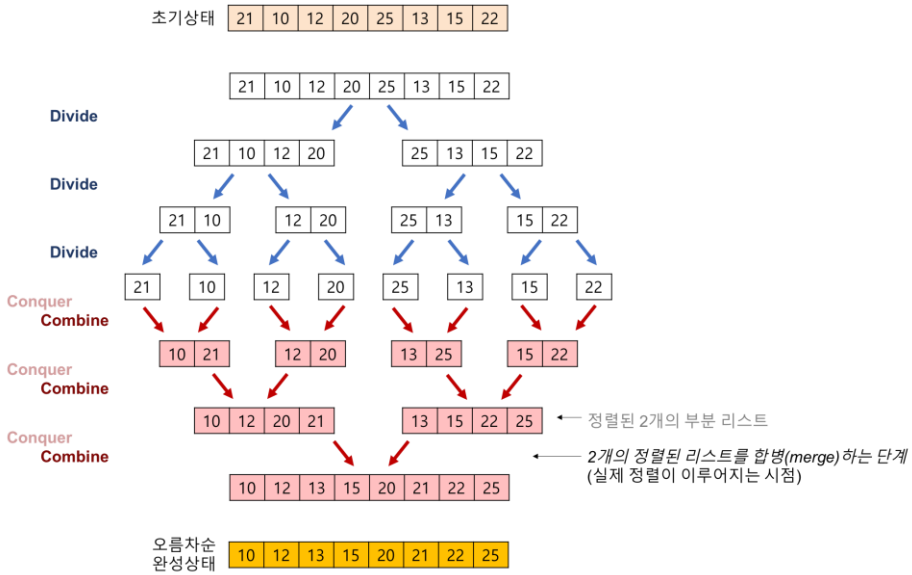
- 정복(Conquer)

  부분 배열을 정렬한다. 부분 배열의 크기가 충분히 작지 않으면 순환 호출을 이용하여 다시 분할 정복 방법을 적용한다.

- 결합(Combine)

  정렬된 부분 배열들을 하나의 배열에 합병한다.

합병 정렬 동작 과정



합병 정렬 알고리즘 예시

배열에 27, 10, 12, 20, 25, 13, 15, 22가 저장되어 있다고 가정하고 자료를 오름차순으로 정렬해봅시다.

2개의 정렬된 리스트를 합병(merge)하는 과정

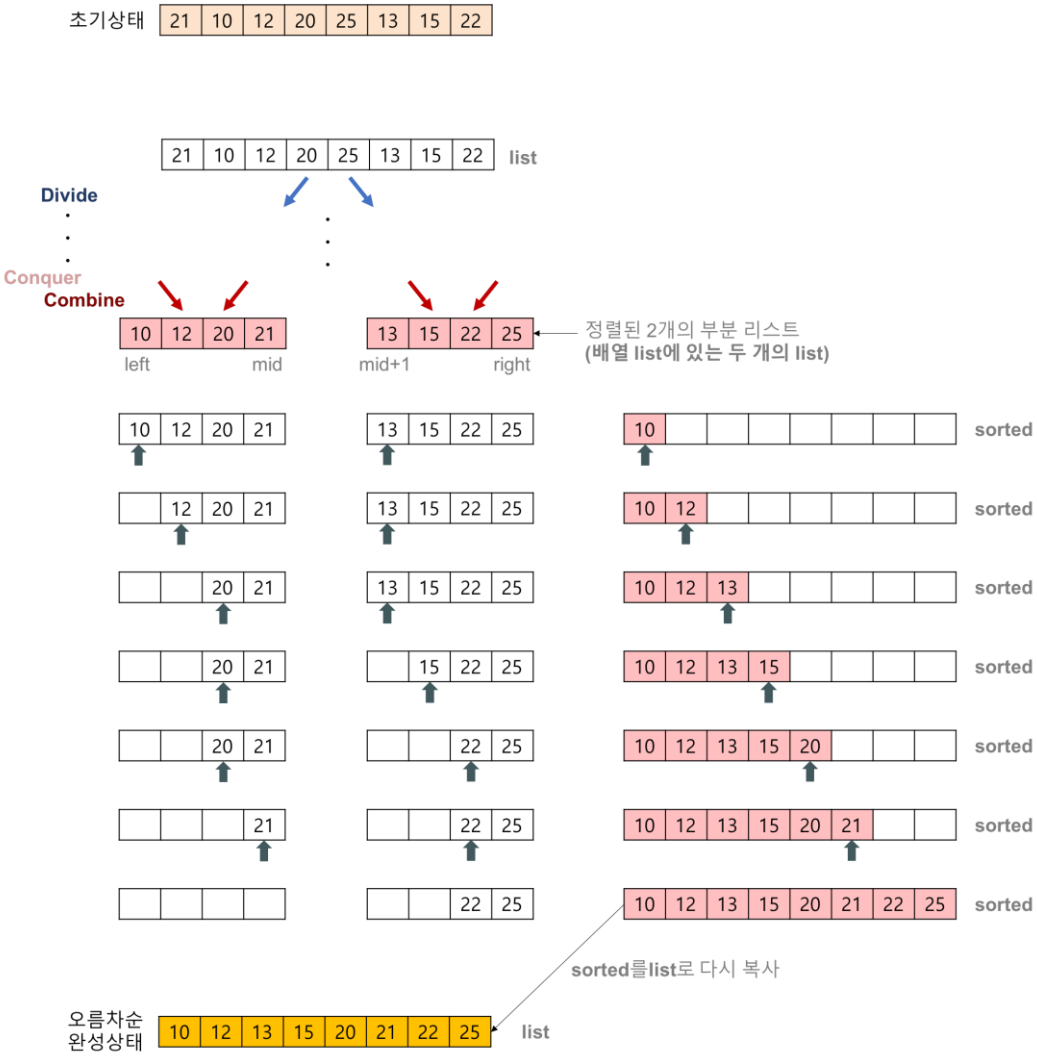
I. 2개의 리스트의 값들을 처음부터 하나씩 비교하여 두 개의 리스트의 값 중 더 작은 값을 새로운 리스트(sorted)옮긴다.

II. 둘 중에서 하나가 끝날 때까지 이 과정을 반복한다.

III. 만약 둘 중에서 하나의 리스트가 먼저 끝나게 되면, 나머지 리스트의 값들을 전부 새로운 리스트(sorted)로 복사한다.

IV. 새로운 리스트(sorted)를 원래의 리스트로 옮긴다.

합병 정렬 중 리스트 합병 예시



합병 정렬 파이썬 코드

```
1  def merge_sort(ary):
2          if len(ary) < 2:
3                  return ary
4
5          mid = len(ary) // 2
6          low_ary = merge_sort(ary[:mid])
7          high_ary = merge_sort(ary[mid:])
8
9          merged_ary = []
10         l = h = 0
11         while l < len(low_ary) and h < len(high_ary):
12                 if low_ary[l] < high_ary[h]:
13                         merged_ary.append(low_ary[l])
14                         l += 1
15                 else:
16                         merged_ary.append(high_ary[h])
17                         h += 1
18         merged_ary += low_ary[l:]
19         merged_ary += high_ary[h:]
20         return merged_ary
```

✓ **2~3행** 길이가 0또는 1인 배열은 이미 정렬된 상태이기 때문에 배열의 길이가 2보다 작다면, 배열 자체를 반환한다.

✓ **5~7행** 배열을 두 부분으로 나누어 각각 merge_sort 함수를 재귀적으로 호출하여 정렬한다. low_ary는 배열의 앞쪽 절반, high_ary는 배열의 뒤쪽 절반을 정렬한 결과이다.

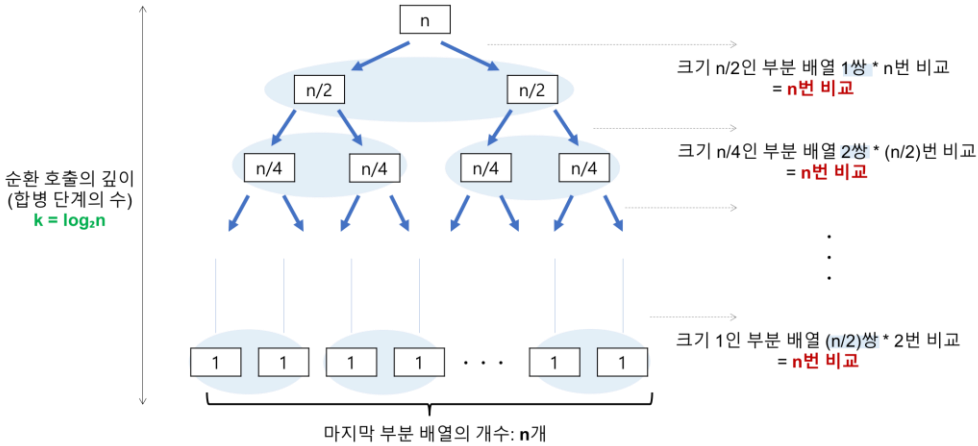
✓ **9~20행** while문에서 두 배열의 요소를 비교하여 merged_ary에 추가한다. while문이 종료된 후, low_ary와 high_ary에 남아있는 요소들을 merged_ary에 병합한다.

합병 정렬 알고리즘의 특징

- 장점
  - 안정적인 정렬 방법으로 데이터의 분포에 영향을 덜 받는다. 즉, 입력 데이터가 무엇이든 간에 정렬되는 시간은 동일하다.
  - 만약 레코드를 연결 리스트(Linked List)로 구성하면, 링크 인덱스만 변경되므로 데이터의 이동은 무시할 수 있을 정도로 작아져서 제자리 정렬(in-place sorting)로 구현할 수 있다.
  - 따라서 크기가 큰 레코드를 정렬할 경우에 연결 리스트를 사용한다면, 합병 정렬은 퀵 정렬을 포함한 다른 어떤 정렬 방법보다 효율적이다.
- 단점
  - 만약 레코드를 배열(Array)로 구성하면, 임시 배열이 필요하다. 그렇기에 제자리 정렬(in-place-sorting)이 아니다.
  - 레코드들의 크기가 큰 경우에는 이동 횟수가 많으므로 매우 큰 시간적 낭비를 초래한다.

합병 정렬의 시간 복잡도

- 분할 단계
  - 비교 연산과 이동 연산이 수행되지 않는다.
- 합병 단계
  - 비교횟수



  - 순환 호출의 깊이 만큼의 합병 단계 x 각 합병 단계의 비교연산

    $= \mathrm{nlog}_2\, n$
  - 이동 횟수
    - 순환 호출의 깊이 만큼의 합병 단계 x 각 합병 단계의 비교연산

      $= 2\,\mathrm{nlog}_2\, n$

$$\boldsymbol{T(n) = nlog_2\, n}\,(\text{비교}) + \boldsymbol{2\,nlog_2\, n}\,(\text{이동}) = \mathbf{3\,nlog_2\, n} = \boldsymbol{O(nlog_2\, n)}$$

03. 퀵 정렬

퀵 정렬(Quick Sort)는 찰스 앤터니 리처드 호어(Charles Antony Richard Hoare)가 개발한 알고리즘이다. 퀵 정렬은 불안정 정렬에 속하며, 다른 원소와의 비교만으로 정렬을 수행할 수 있는 비교 정렬에 속한다.

분할 정복 알고리즘의 하나로, 평균적으로 매우 빠른 수행 속도를 자랑한다. 합병 정렬(merge sort)과 달리 퀵 정렬은 리스트를 비균등하게 분할한다.

퀵 정렬 과정

하나의 리스트를 피벗(pivot)을 기준으로 두 개의 비균등한 크기로 분할하고, 분할된 리스트를 정렬한 다음, 두 개의 정렬된 부분 리스트를 합하여 전체가 정렬된 리스트가 되게 하는 방법이다.

- 분할(Divide)

  입력 배열을 피벗을 기준으로 비균등하게 2개의 부분 배열(피벗을 중심으로 왼쪽(피벗보다 작은 요소들), 오른쪽(피벗보다 큰 요소들)로 분할한다.

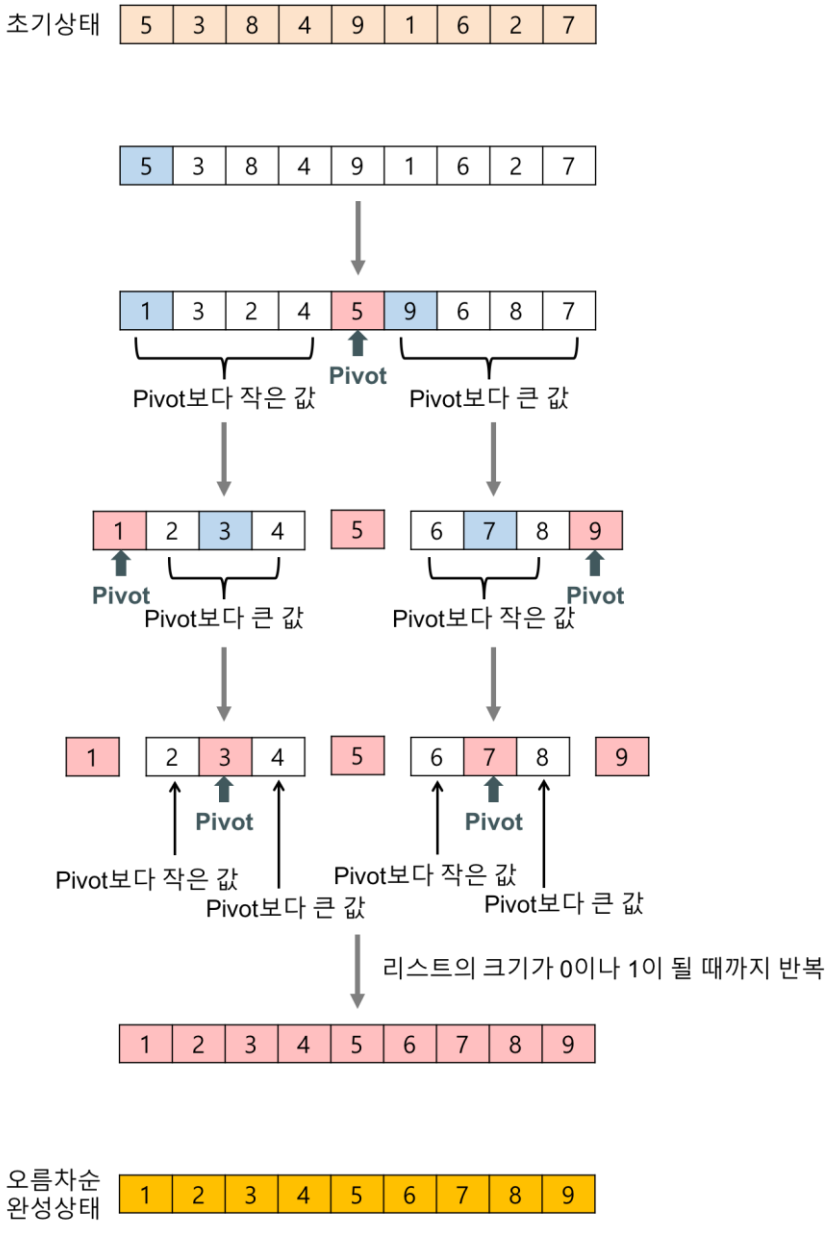
- 정복(Conquer)

  부분 배열을 정렬한다. 부분 배열의 크기가 충분히 작지 않으면 순환 호출을 이용하여 다시 분할 정복 방법을 적용한다.

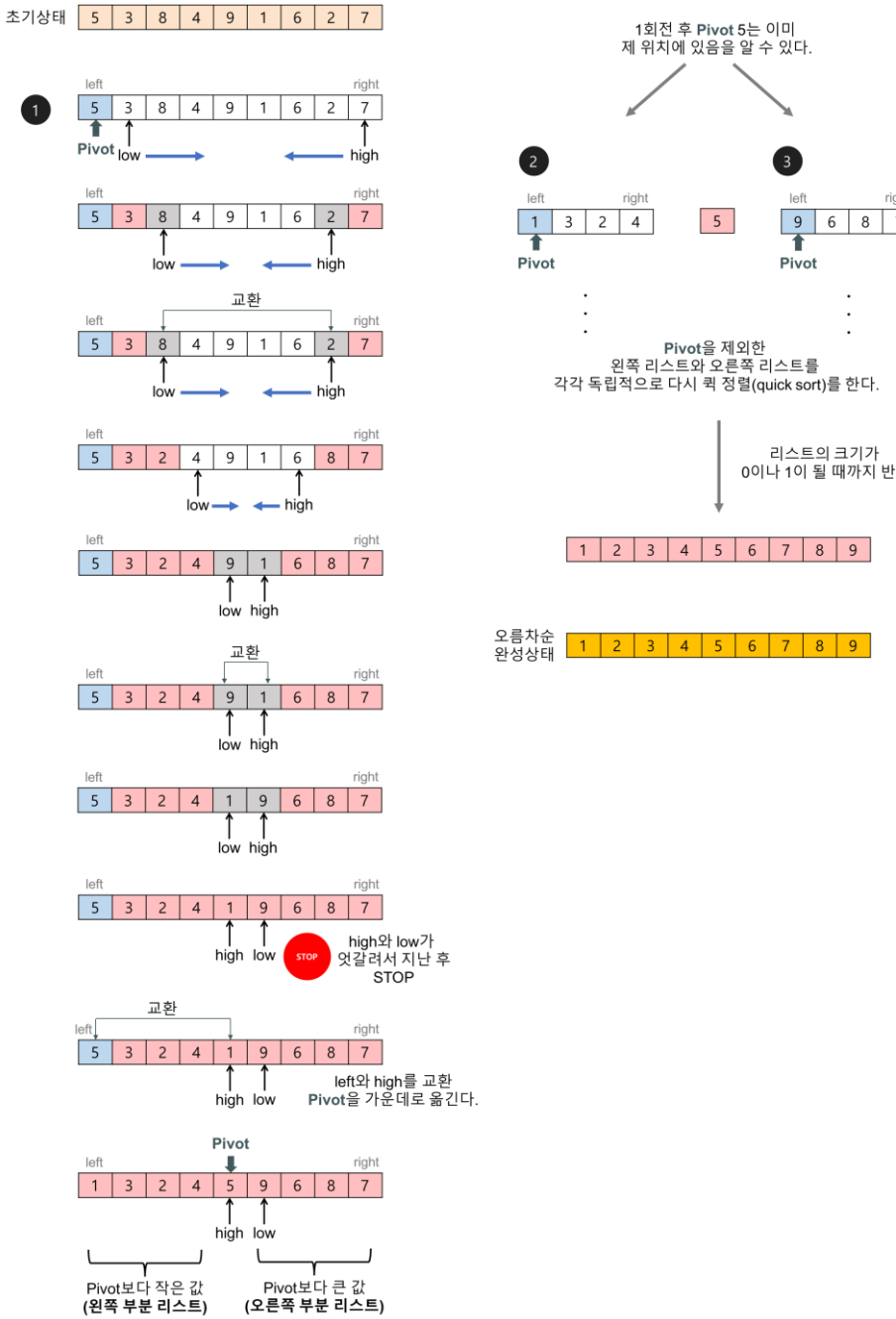
- 결합(Combine)

  정렬된 부분 배열들을 하나의 배열에 합병한다.

퀵 정렬 동작 과정


초기상태
5 3 8 4 9 1 6 2 7
5 3 8 4 9 1 6 2 7
1 3 2 4 5 9 6 8 7
Pivot
Pivot보다 작은 값
Pivot보다 큰 값
1 2 3 4
5
6 7 8 9
Pivot
Pivot보다 큰 값
Pivot보다 작은 값
Pivot
1
2 3 4
5
6 7 8
9
Pivot
Pivot
Pivot보다 작은 값
Pivot보다 큰 값
Pivot보다 작은 값
Pivot보다 큰 값
리스트의 크기가 0이나 1이 될 때까지 반복
1 2 3 4 5 6 7 8 9
오름차순
완성상태
1 2 3 4 5 6 7 8 9

퀵 정렬 알고리즘 예시


초기상태
5 3 8 4 9 1 6 2 7
1
left
right
Pivot
low
high
교환
STOP
high와 low가 엇갈려서 지난 후 STOP
left와 high를 교환
Pivot을 가운데로 옮긴다.
Pivot
Pivot보다 작은 값
(왼쪽 부분 리스트)
Pivot보다 큰 값
(오른쪽 부분 리스트)
1회전 후 Pivot 5는 이미
제 위치에 있음을 알 수 있다.
2
3
1 3 2 4
5
9 6 8 7
Pivot을 제외한
왼쪽 리스트와 오른쪽 리스트를
각각 독립적으로 다시 퀵 정렬(quick sort)를 한다.
리스트의 크기가
0이나 1이 될 때까지 반복
1 2 3 4 5 6 7 8 9
오름차순
완성상태

퀵 정렬 알고리즘 예시

- 피벗 값을 입력 리스트의 첫 번째 데이터로 정하는 상황을 고려하자.

- 2개의 인덱스 변수(low, high)를 이용해서 리스트를 두 개의 부분 리스트로 나눈다.

- 1회전: 피벗이 5인 경우
    I. low는 왼쪽에서 오른쪽으로 탐색해가다가 피벗보다 큰 데이터(8)을 찾으면 멈춘다.
    II. high는 오른쪽에서 왼쪽으로 탐색해가다가 피벗보다 작은 데이터(2)를 찾으면 멈춘다.
    III. Low와 high가 가리키는 두 데이터를 서로 교환한다.
    IV. 이 탐색-교환 과정은 low와 high가 엇갈릴 때까지 반복한다.

- 2회전: 피벗(1회전의 왼쪽 부분 리스트의 첫 번째 데이터)이 1인 경우
    - 위와 동일한 방법으로 반복한다.

- 3회전: 피벗(1회전의 오른쪽 부분 리스트의 첫 번째 데이터)이 9인 경우
    - 위와 동일한 방법으로 반복한다.

퀵 정렬 파이썬 코드

```
1  def quick_sort(ary, start, end):
2          if start >= end:
3                  return
4          pivot = start
5          left = start+1
6          right = end
7
8          while left <= right:
9                  while left <= end and ary[left] <= ary[pivot]:
10                         left+=1
11                 while right > start and ary[right] >= ary[pivot]:
12                         right-=1
13                 if left>right:
14                         ary[right], ary[pivot] = ary[pivot], ary[right]
15                 else:
16                         ary[left], ary[right] = ary[right], ary[left]
17
18         quick_sort(ary, start, right-1)
19         quick_sort(ary, right+1, end)
20
21 quick_sort(ary, 0, len(ary)-1)
```

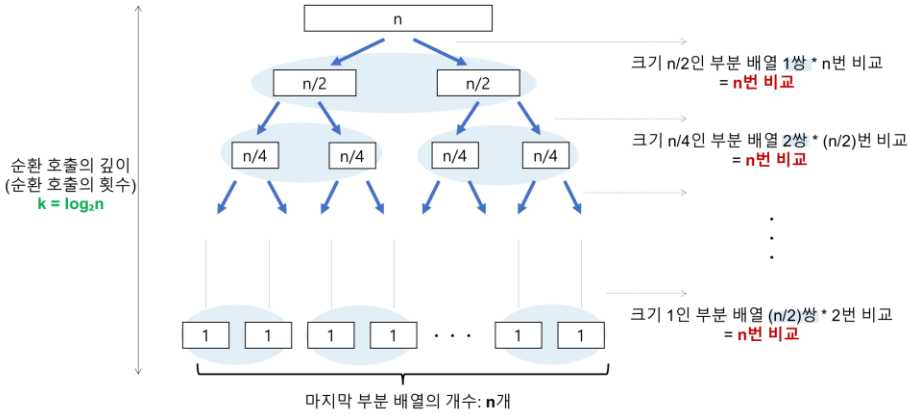
- ✓ **2~3행** 길이가 0또는 1인 배열은 이미 정렬된 상태이기 때문에 배열의 길이가 2보다 작다면, 배열 자체를 반환한다.
- ✓ **4행** 피벗의 초기값을 배열의 첫 번째 요소로 선택한다
- ✓ **9~10행** 피벗보다 큰 데이터를 찾을 때까지 반복한다.
- ✓ **11~12행** 피벗보다 작은 데이터를 찾을 때까지 반복한다.
- ✓ **13~14행** 엇갈렸다면, 작은 데이터와 피벗을 교체한다
- ✓ **15~16행** 엇갈리지 않았다면, 작은 데이터와 큰 데이터를 교체한다.
- ✓ **18~19행** 분할 이후 왼쪽 부분과 오른쪽 부분에서 각각 정렬을 수행한다.

퀵 정렬 알고리즘의 특징

- 장점
  - 속도가 빠르다. 시간 복잡도가 $O(\text{nlog}_2 n)$을 가지는 다른 정렬 알고리즘과 비교했을 때도 가장 빠르다.
  - 추가 메모리 공간을 필요로 하지 않는다. 퀵 정렬은 $O(\log n)$만큼의 메모리를 필요로 한다.
- 단점
  - 정렬된 리스트에 대해서는 퀵 정렬의 불균형 분할에 의해 오히려 수행시간이 더 많이 걸린다.

퀵 정렬의 시간 복잡도

- **최선의 경우**
  - 비교횟수

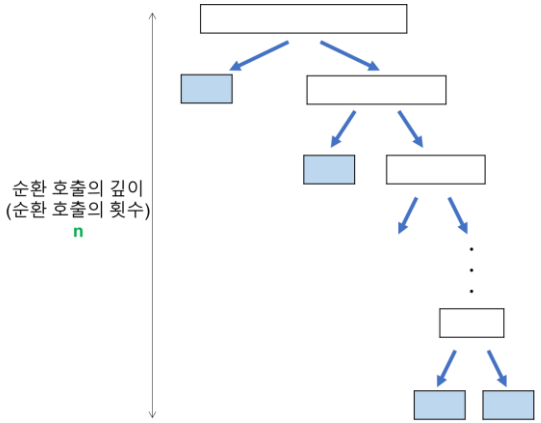


    - 순환 호출의 깊이 x 각 순환 호출 단계의 비교 연산 = $\mathrm{nlog}_2\, n$
  - 이동 횟수
    - 비교 횟수보다 적으므로 무시할 수 있다.

최선의 경우: $T(n) = O(\mathrm{nlog}_2\, n)$

퀵 정렬의 시간 복잡도

- 최악의 경우
  - 리스트가 계속 불균형하게 나누어 지는 경우
  - 이미 정렬된 리스트에 대하여 퀵 정렬을 실행하는 경우



  - 비교횟수
    - 순환 호출의 깊이 x 각 순환 호출 단계의 비교 연산 = n^2
  - 이동 횟수
    - 비교 횟수보다 적으므로 무시할 수 있다.

최악의 경우: $T(n) = O(n^2)$

정렬 알고리즘 시간 복잡도 비교

알고리즘	Best	Avg	Worst
삽입 정렬	$O(n)$	$O(n^2)$	$O(n^2)$
선택 정렬	$O(n^2)$	$O(n^2)$	$O(n^2)$
버블 정렬	$O(n^2)$	$O(n^2)$	$O(n^2)$
합병 정렬	$O(\text{nlog}_2\, n)$	$O(\text{nlog}_2\, n)$	$O(\text{nlog}_2\, n)$
퀵 정렬	$O(\text{nlog}_2\, n)$	$O(\text{nlog}_2\, n)$	$O(n^2)$

- **단순(구현이 간단)하지만 비효율적인 방법**
  - 삽입 정렬, 선택 정렬, 버블 정렬

- **복잡하지만 효율적인 방법**
  - 퀵 정렬, 합병 정렬

제 7 장

특수 정렬 알고리즘

1) 계수 정렬
2) 힙 정렬
3) 위상 정렬

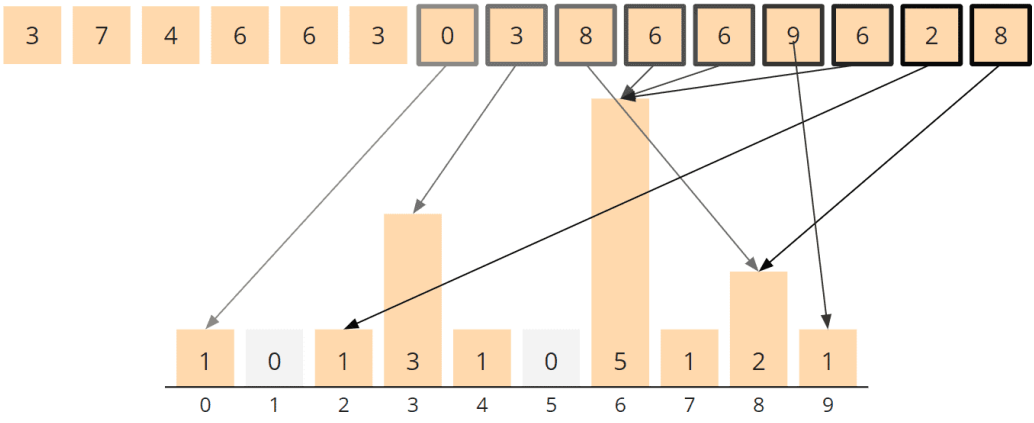
01. 계수 정렬

계수 정렬(Counting Sort)는 데이터 값을 직접 비교하지 않고, 단순하게 배열에 존재하는 수의 개수를 세어주고, 이 정보를 바탕으로 정렬을 수행한다. 그리고 여기서 수와 수의 개수는 배열 혹은 딕셔너리의 인덱스와 해당 값을 통해서 기록할 수 있다.

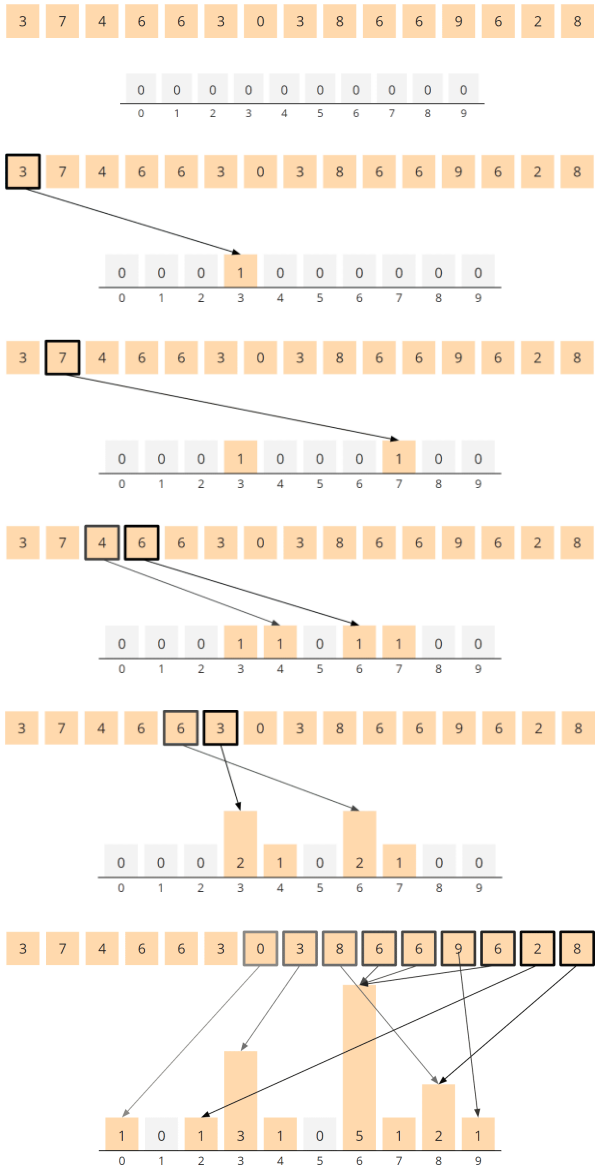
정렬은 Comparisons 방식과 Non-Comparisons 방식으로 나뉘는데, 계수 정렬은 비비교 정렬(Non-Comparison) 정렬에 해당하여 정렬할 데이터에 대한 사전 지식을 이용한다.

계수 정렬 과정

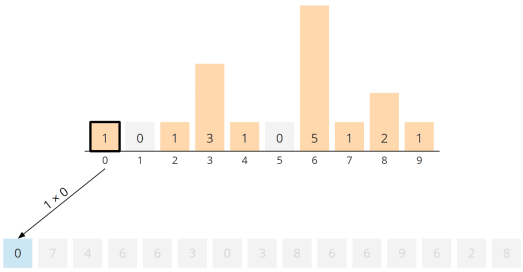
1. 정렬하고자 하는 배열의 최댓값을 구한다.
2. 최댓값 크기의 배열에 각 원소를 순회하며 해당 값이 몇 개인지 저장한다.
3. 저장된 데이터를 순서대로 출력한다.

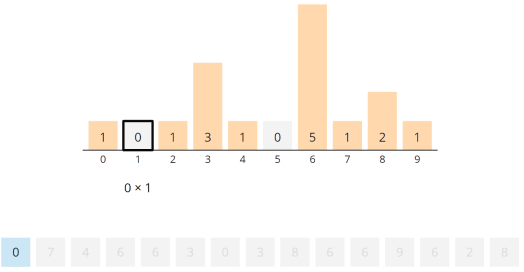


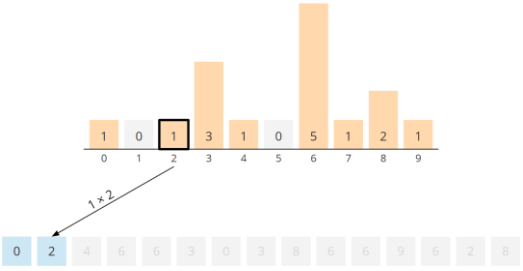
계수 정렬 동작 과정

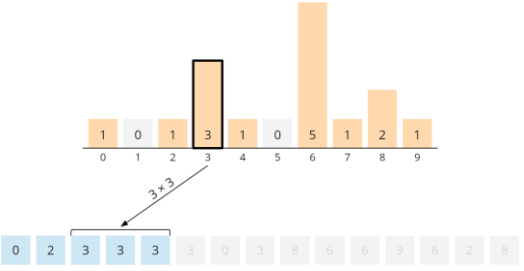

3 7 4 6 6 3 0 3 8 6 6 9 6 2 8
0 0 0 0 0 0 0 0 0 0
0 1 2 3 4 5 6 7 8 9
3 7 4 6 6 3 0 3 8 6 6 9 6 2 8
0 0 0 1 0 0 0 0 0 0
0 1 2 3 4 5 6 7 8 9
3 7 4 6 6 3 0 3 8 6 6 9 6 2 8
0 0 0 1 0 0 0 1 0 0
0 1 2 3 4 5 6 7 8 9
3 7 4 6 6 3 0 3 8 6 6 9 6 2 8
0 0 0 1 1 0 1 1 0 0
0 1 2 3 4 5 6 7 8 9
3 7 4 6 6 3 0 3 8 6 6 9 6 2 8
0 0 0 2 1 0 2 1 0 0
0 1 2 3 4 5 6 7 8 9
3 7 4 6 6 3 0 3 8 6 6 9 6 2 8
1 0 1 3 1 0 5 1 2 1
0 1 2 3 4 5 6 7 8 9

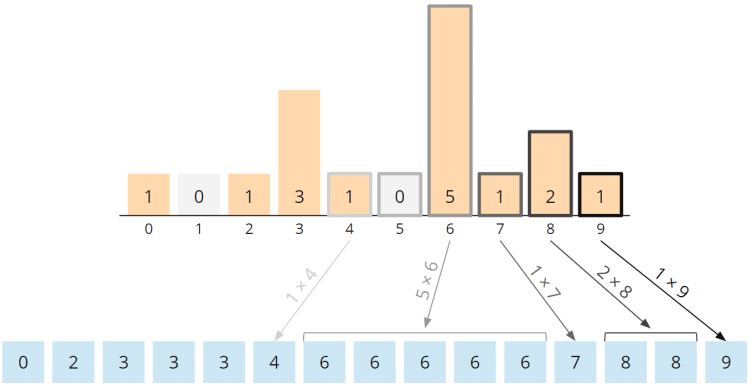
계수 정렬 동작 과정


1 0 1 3 1 0 5 1 2 1
0 1 2 3 4 5 6 7 8 9
1×0
0 7 4 6 6 3 0 3 8 6 6 9 6 2 8


1 0 1 3 1 0 5 1 2 1
0 1 2 3 4 5 6 7 8 9
0×1
0 7 4 6 6 3 0 3 8 6 6 9 6 2 8


1 0 1 3 1 0 5 1 2 1
0 1 2 3 4 5 6 7 8 9
1×2
0 2 4 6 6 3 0 3 8 6 6 9 6 2 8


1 0 1 3 1 0 5 1 2 1
0 1 2 3 4 5 6 7 8 9
3×3
0 2 3 3 3 3 0 3 8 6 6 9 6 2 8


1 0 1 3 1 0 5 1 2 1
0 1 2 3 4 5 6 7 8 9
1×4
5×6
1×7
2×8
1×9
0 2 3 3 3 4 6 6 6 6 6 7 8 8 9

7-1. 리스트 자료형을 활용한 계수 정렬 파이썬 코드

```
1  def counting_sort(ary):
2      max_ary = max(ary)
3      count = [0] * (max_ary + 1)
4      sorted_ary = list()
5
6      for i in ary:
7          count[i] += 1
8
9      for i in range(max_ary + 1):
10         for _ in range(count[i]):
11             sorted_ary.append(i)
12     return sorted_ary
```

7-2. 딕셔너리 자료형을 활용한 계수 정렬 파이썬 코드

```
1  def counting_sort(ary):
2      count = dict()
3      sorted_ary = list()
4
5      for i in ary:
6          if i in count:
7              count[i] += 1
8          else:
9              count[i] = 1
10
11     for i in sorted(count.keys()):
12         for _ in range(count[i]):
13             sorted_ary.append(i)
14     return sorted_ary
```

계수 정렬 알고리즘의 특징

- **장점**
  - 계수 정렬은 배열에 있는 원소들의 개수만 세고 작은 수부터 나열하므로 시간 복잡도는 $O(n+k)$라고 할 수 있다.
  - 이 때 k는 배열에 있는 최댓값이다.
  - 공간 복잡도는 k만큼의 배열을 추가로 만들어야 하기 때문에 $O(k)$라고 할 수 있다.

- **단점**
  - 배열의 인덱스는 양수만 존재하기에 데이터(값)은 양수여야 한다.
  - 값의 범위가 너무 크지 않아야 한다. (메모리 사이즈를 넘어서는 안된다.)

02. 힙 정렬

힙 정렬

힙 정렬(Heap Sort)이란 병합 정렬(Merge Sort)와 퀵 정렬(Quick Sort)만큼 빠른 정렬 알고리즘이다. 힙 정렬은 최댓값이나 최솟값을 빠르게 검색하기 위한 우선순위 큐를 위하여 만들어진 자료구조인 힙 트리 구조(Heap Tree Structure)을 이용하는 정렬 방법인데, 이를 알기 위해서는 사전 지식들이 필요하다.

트리(Tree)

- 계층적 구조를 표현하는 자료구조

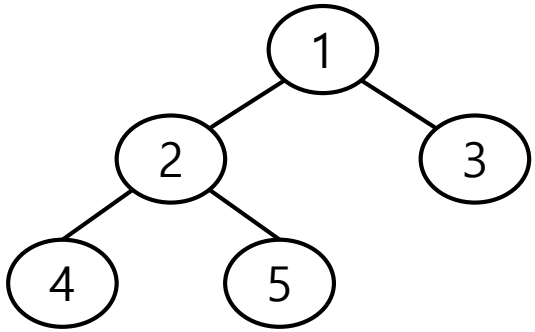
이진 트리(Binary Tree)

- 트리에 있는 모든 노드들을 봤을 때, 각 노드가 최대 두 개의 자식을 갖는 트리

완전 이진 트리(Complete Binary Tree)

- 마지막 레벨을 제외하고 모든 레벨이 완전히 채워져 있는 트리
- 마지막 레벨은 노드가 왼쪽에서 오른쪽으로 채워져야 함
- 배열을 이용하여 표현 가능

완전 이진트리를 생성할 때는 데이터가 루트(Root) 노드부터 시작하여 자식 노드가 왼쪽 자식 노드, 오른쪽 자식 노드로 차근차근 들어간다.

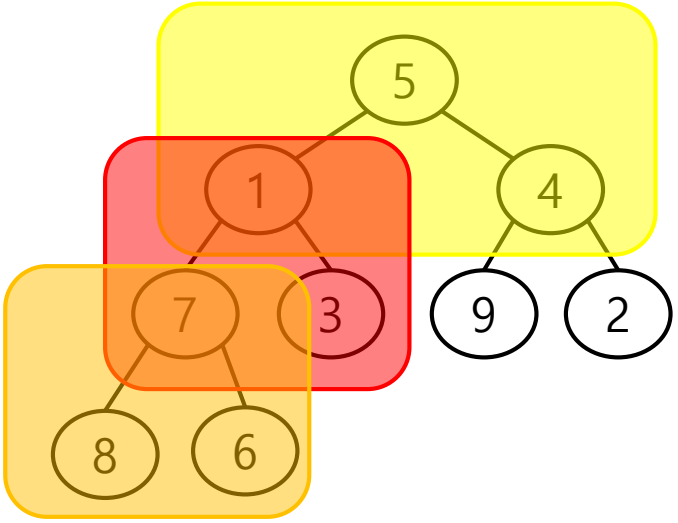


그렇기 때문에, 위 그림과 같이 번호 순서대로 노드들이 추가된다. 완전 이진트리는 모든 레벨이 완전히 채워져 있거나, 마지막 레벨이 왼쪽부터 채워져 있기에 트리의 노드를 배열에 연속적으로 배치할 수 있기 때문에 배열로 표현이 가능하다.

index	0	1	2	3	4
data	1	2	3	4	5

완전이진트리가 생성되는 순서대로 5개의 노드를 0번 인덱스부터 4번 인덱스까지 순서대로 입력하여 생성된 표는 위와 같다.
이런식으로 우리는 파이썬의 리스트를 사용하여 완전 이진트리를 배열을 이용하여 표현할 수 있다.

아래 완전이진트리를 배열을 다시 표현을 해보자.



index	0	1	2	3	4	5	6	7	8
data	5	1	4	7	3	9	2	8	6

노란색 네모칸을 보면 부모 노드는 0번 인덱스의 5, 왼쪽 자식 노드는 1번 인덱스인 1, 오른쪽 자식 노드는 2번 인덱스의 4이다.

빨간색 네모칸을 보면 부모 노드는 1번 인덱스의 1, 왼쪽 자식 노드는 3번 인덱스의 7, 오른쪽 자식 노드는 4번 인덱스의 3이다.

주황색 네모칸을 보면 부모 노드는 3번 인덱스의 7, 왼쪽 자식 노드는 7번 인덱스의 8, 오른쪽 자식 노드는 8번 인덱스의 6이다.

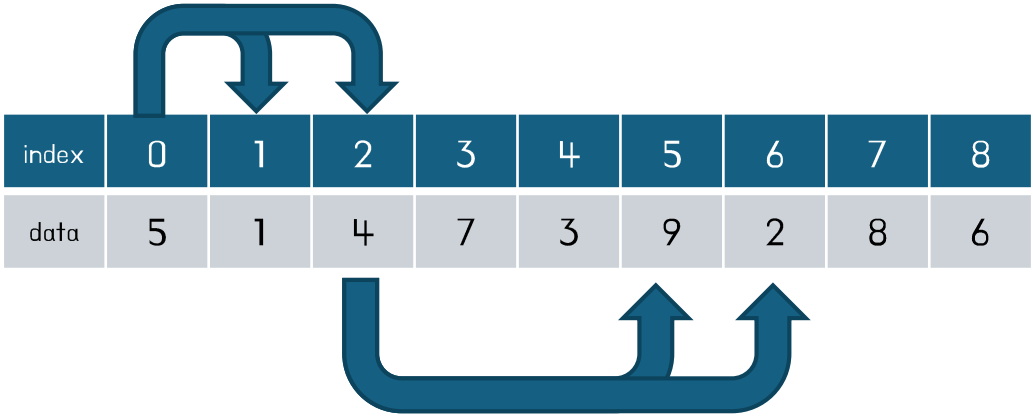
이 규칙을 토대로 보면, 부모 인덱스를 i라고 하였을 때 왼쪽 자식 노드의 인덱스는 $2 * i + 1$, 오른쪽 자식 노드의 인덱스는 $2 * i + 2$임을 알 수 있다.

부모 노드 : i

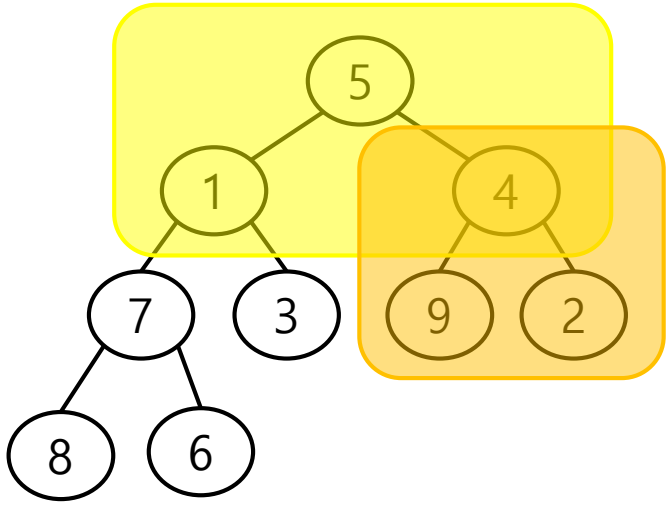
왼쪽 자식 노드 : $2 * i + 1$

오른쪽 자식 노드 : $2 * i + 2$

우리는 위와 같은 사실을 이용해서 배열을 통해 원하는 노드에 접근할 수 있다.



index	0	1	2	3	4	5	6	7	8
data	5	1	4	7	3	9	2	8	6

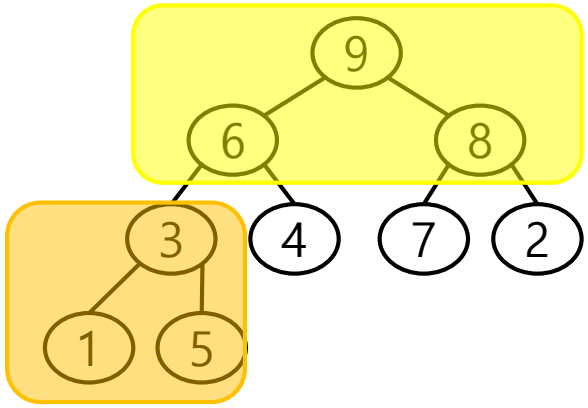


힙(Heap)

- 우선순위 큐를 위하여 만들어진 자료구조
- 최댓값이나 최솟값을 빠르게 검색 가능
- 완전 이진 트리의 일종
- 최대 힙(max heap property)과 최소 힙(min heap property) 존재
- 반 정렬 상태(느슨한 정렬 상태) 유지

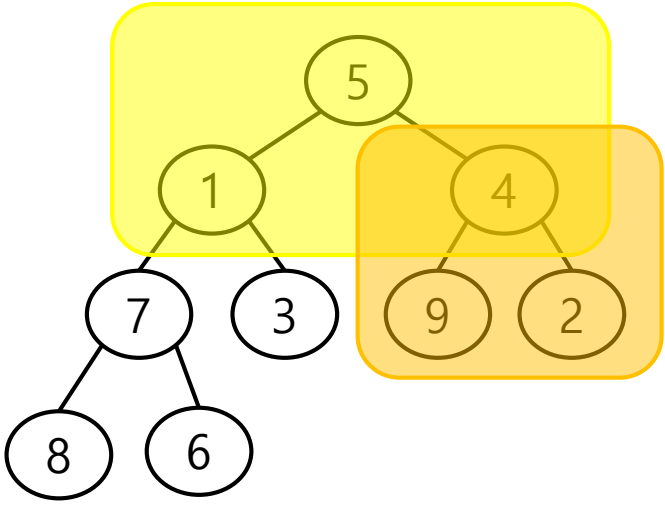
최대 힙(Max Heap)

- 부모 노드가 자식 노드보다 큰 힙
- 루트 노드가 최댓값을 가지며, 서브트리의 루트 노드는 해당 서브트리의 최댓값을 가짐



위 그림을 보면, 어느 노드를 보아도 부모 노드가 항상 자식 노드보다 크다는 것을 알 수 있다. 이러한 구조를 최대 힙이라고 한다.

다만, 트리(Tree) 안에서 특정 노드로 인하여 최대 힙이 붕괴되는 경우가 있다.



노란색을 보면, 부모 노드가 두 자식 노드보다 값이 크기에 최대 힙을 만족하지만,

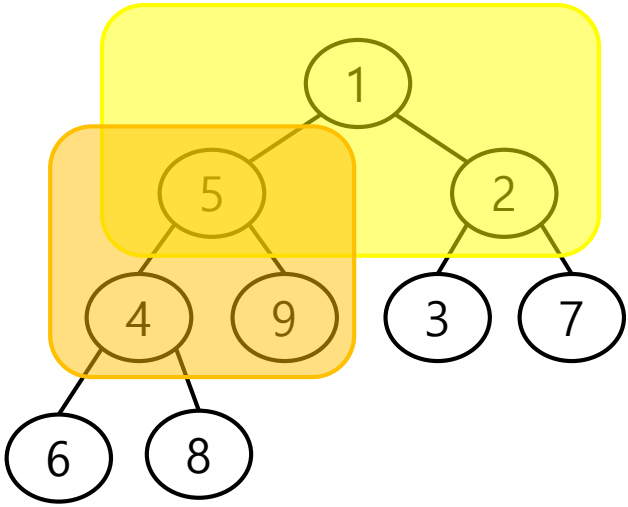
주황색을 보면, 부모 노드가 두 자식 노드 보다 크지 않기에 최대 힙이 아니게 된다.

최소 힙(Min Heap)

부모 노드가 자식 노드보다 작은 힙

루트 노드가 최솟값을 가짐

서브트리의 루트 노드는 해당 서브트리의 최솟값을 가짐



최대 힙과는 반대로 위 그림을 보면, 어느 노드를 보아도 부모 노드가 항상 자식 노드보다 작다는 것을 알 수 있다. 이러한 구조를 최소 힙이라고 한다.

최대 힙과 마찬 가지로, 트리(Tree) 안에서 특정 노드로 인하여 최소 힙이 붕괴되는 경우가 있다.

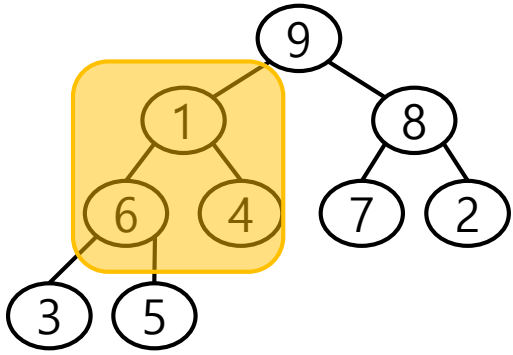
노란색 영역을 보면 부모 노드가 두 자식 노드보다 작기에 최소 힙을 만족하지만, 주황색 영역을 보면 부모 노드가 왼쪽 자식 노드 보다 값이 크기 때문에 최소 힙을 만족하지 않는다.

지금까지 힙 정렬을 학습하기 위한 관련 사전 지식들을 알아보았다.

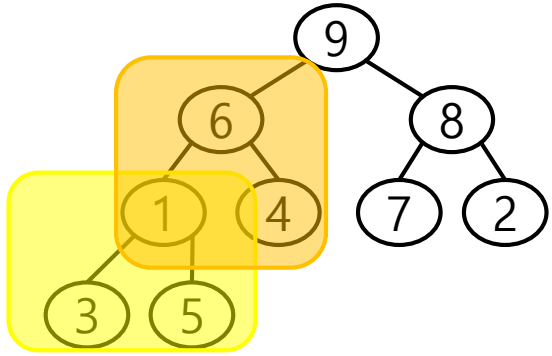
완전 이진트리의 일종인 힙에는 최대 힙과 최소 힙이 있고, 중간 노드 부분에서 최대 힙/최소 힙을 만족하지 않게 되면 최대 힙/최소 힙이 아니다라고 할 수 있다.

힙 생성 알고리즘(Heapify Algorithm)

힙 정렬을 수행하기 위해서는 힙 생성 알고리즘(Heapify Alogrithm)을 사용하며, 힙 정렬을 구현하기 위해 필요한 핵심 알고리즘이다. 최대 힙 기준으로 적용할 때, 특정한 노드의 두 자식 중에서 더 큰 자식과 자신의 위치를 바꾸는 알고리즘이다. 이 힙 생성 알고리즘은 특정한 하나의 노드에 대해서 수행하기에, 하나의 노드를 제외하고는 최대 힙이 구성되어 있는 상태라고 가정을 하고 진행한다는 특징이 있다.

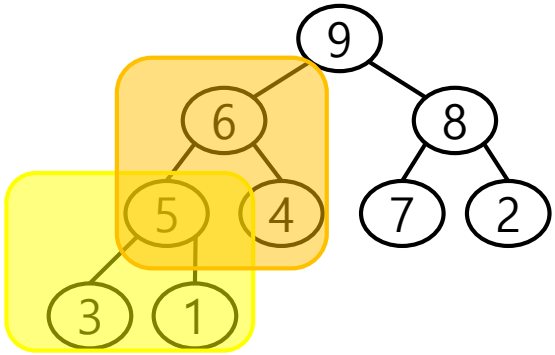


위 주황색 영역은 최대 힙 구조를 붕괴하므로, 6과 4 두 자식 노드 중 더 큰 왼쪽 자식 노드를 부모 노드인 1과 바꿔주면 해당 구조는 최대 힙 구조를 만족하게 된다.



이 과정을 자식이 존재하지 않을 때 까지 반복해주면 된다.

노란색 영역을 마저 처리하게 되면, 3과 5중 더 큰 오른쪽 자식 노드를 부모 노드인 1과 바꾸어 주면 아래와 같이 최대 힙 구조를 만족하게 된다.



최종적으로 저희가 바꾸었던 영역을 제외하고 나머지 부분은 이미 최대 힙 구조가 되어있다고 가정을 하였기 때문에, 주황색 영역을 최대 힙 구조로 변경하여 주면 위 트리는 최종적으로 최대 힙 구조를 가지게 되었다.

이 Max-Heapify 코드를 작성하면 다음과 같다.

7-3. 특정 노드를 루트 노드로 하여 최대 힙 만들기

```
def max_heapify(A, i, n):
        largest = i # 현재 노드를 가장 큰 값으로 가정
        left = 2 * i + 1 # 왼쪽 자식 노드의 인덱스
        right = 2 * i + 2 # 오른쪽 자식 노드의 인덱스


        # 왼쪽 자식 노드가 존재하고, 현재 노드보다 큰 경우
        if left < n and A[left] > A[largest]:
                largest = left

        # 오른쪽 자식 노드가 존재하고, 현재 노드보다 큰 경우
        if right < n and A[right] > A[largest]:
                largest = right

        # 가장 큰 값이 현재 노드가 아닌 경우
        if largest != i:
                A[i], A[largest] = A[largest], A[i] # 현재 노드와 가장 큰 자식
                노드를 교환

                # 교환 후 영향을 받은 서브트리를 재귀적으로 힙화
                max_heapify(A, largest, n)


# 사용 예시:
A = [9, 1, 8, 6, 4, 7, 2, 3, 5]
i = 1
n = len(A)
max_heapify(A, i, n)
print(A)
```

7-3. 출력 결과

```
[9, 6, 8, 5, 4, 7, 2, 3, 1]
```

2~4행

왼쪽 자식 노드, 오른쪽 자식 노드를 리스트를 이용한 인덱스를 통하여 설정한다. 그리고, 현재 부모 노드가 가장 큰 값이라고 가정한다.

7~9행

왼쪽 자식 노드와 부모 노드의 값을 비교해서, 왼쪽 자식 노드가 크다면, largest 값을 왼쪽 자식 노드의 값으로 변경한다.

11~13행

오른쪽 자식 노드와 부모 노드의 값을 비교해서, 오른쪽 자식 노드가 크다면, largest 값을 오른쪽 자식 노드의 값으로 변경한다.

15~19행

7~13행을 통해서 왼쪽 자식 노드, 오른쪽 자식 노드, 부모 노드 중에서 가장 큰 값이 결정이 되었고, 이를 토대로 자식 노드 중 가장 큰 값을 부모 노드와 변경하고, 부모 노드가 크다면 현재 상태 그대로 유지한다.

20~21행

영향을 받은 노드를 재귀적으로 방금까지의 과정을 반복한다.

24~29행

최대 힙 알고리즘을 실행한다.

밤금 max_heapify 함수를 사용하게 되면, 특정 노드 위치로부터 최대 힙 알고리즘을 수행한다. 전체 노드가 최대 힙을 만족시키게 하기 위해서 우리는 다음과 같이 build_max_heap 함수를 만들어 보도록 하겠다.

7-4. 전체 노드 최대 힙 만들기

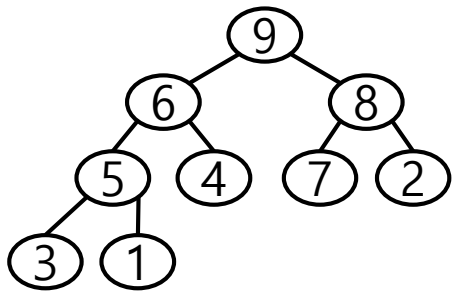
```
1  def build_max_heap(A):
2          n = len(A)
3          # 배열의 중간부터 시작하여 첫 번째 요소까지 반복
4          for i in range(n // 2 - 1, -1, -1):
5          max_heapify(A, i, n)
6
7
8  # 사용 예시:
9  A = [9, 1, 8, 6, 4, 7, 2, 3, 5]
10 build_max_heap(A)
11 print(A)
```

7-4. 출력 결과

```
[9, 6, 8, 5, 4, 7, 2, 3, 1]
```

1~5행

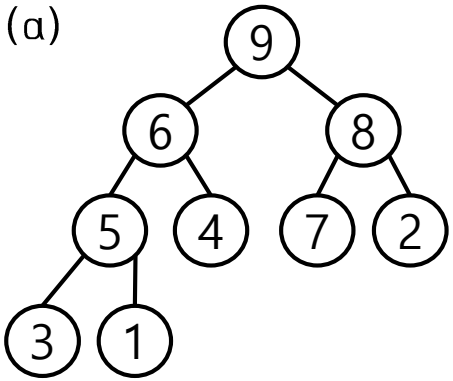
전체 노드의 절반 노드부터 루트 노드까지 거꾸로 돌아가면서 max_heapify 실행

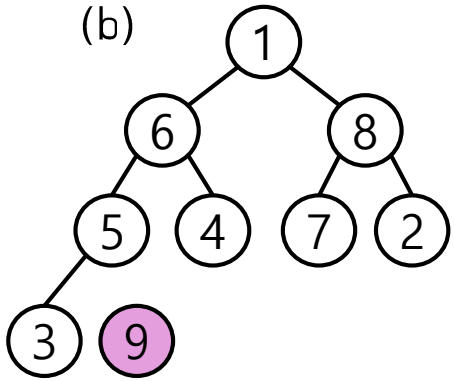


이 코드를 실행하면 최대 힙을 만족시킬 수 있다.

힙 정렬(Heap Sort)

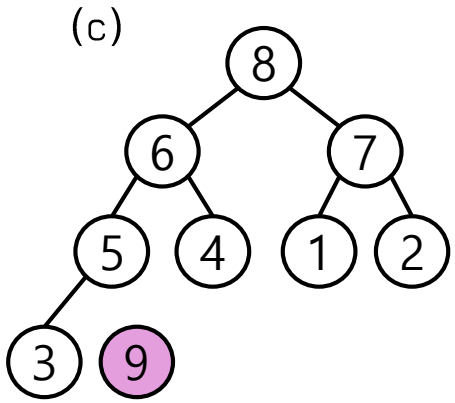
지금까지 우리가 배운 최대 힙과, 힙 생성 알고리즘을 이용하여 힙 정렬을 구현할 수 있다.





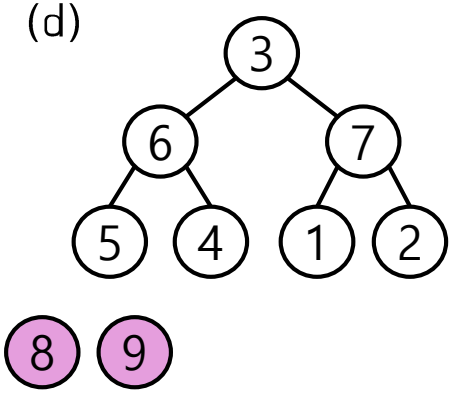
제일 처음 (a)와 같이 최대 힙을 이루고 있는 힙을 준비한다. 이 힙은 힙 생성 알고리즘을 통하여 만들 수 있다. 루트 노드와, 가장 마지막 노드(마지막 인덱스)의 값을 가장 처음에 바꿔준다. (b)와 같이 루트 노드 였던 9는 마지막 노드에 위치하게 되고, 해당 위치에 있던 1은 루트 노드로 이동하게 된 것을 확인할 수 있다.

그 다음 (b)에서 힙 생성 알고리즘을
루트 노드를 시작으로 맨 마지막 노드 -1 번
위치까지 실행한다.
마지막 노드를 제외하고 (c)와 같이
최대 힙이 만들어진 것을 확인할 수 있다.

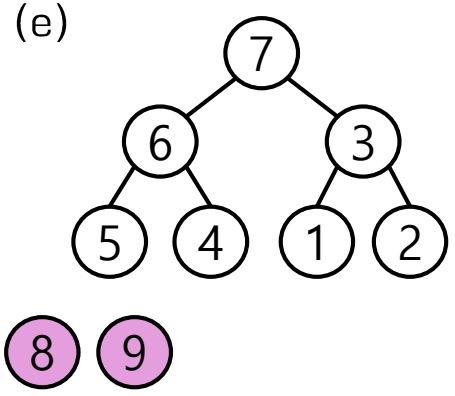


(b)~(c)의 과정을 트리의 크기를 1씩 줄여가면서 계속 반복한다.

(d)

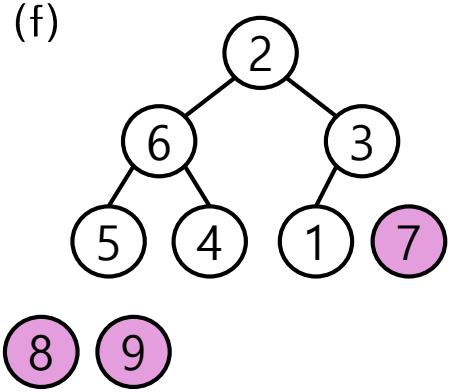


(e)

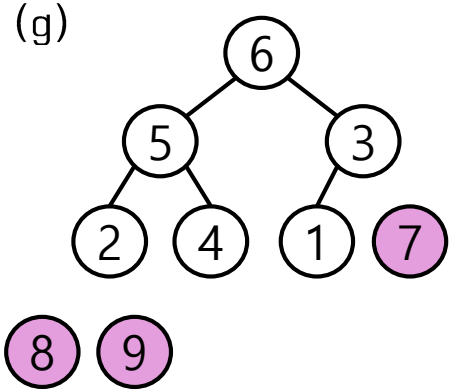


루트 노드였던 8과 마지막 노드 3을 교환 한 후에, 힙 생성 알고리즘을 수행했다.

(f)

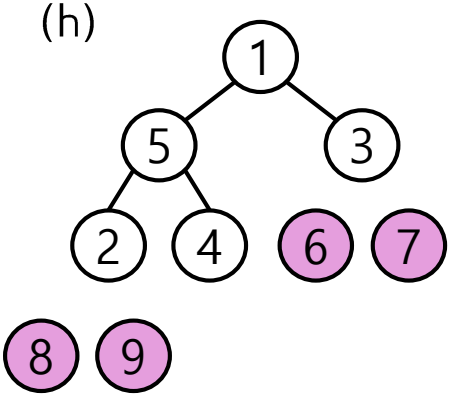


(g)

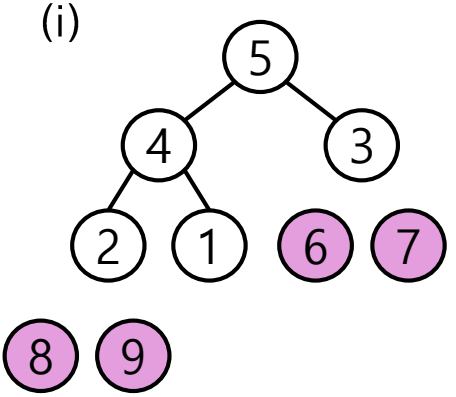


루트 노드였던 3과 마지막 노드 2를 교환 한 후에, 힙 생성 알고리즘을 수행했다.

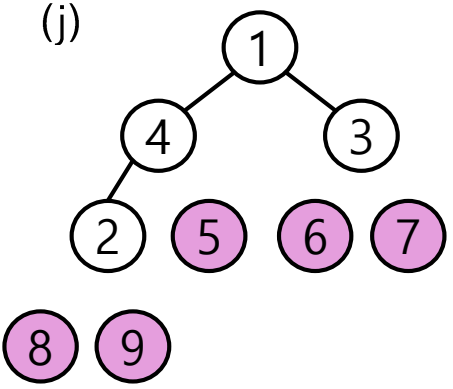
(h)


1
5
3
2
4
6
7
8
9

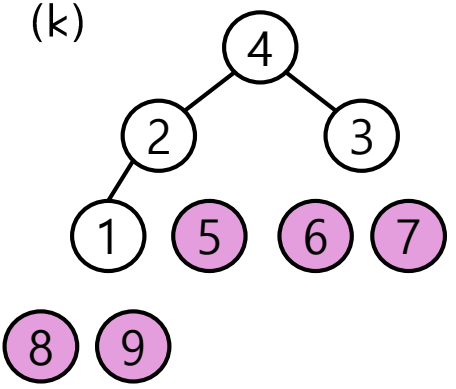
(i)


5
4
3
2
1
6
7
8
9

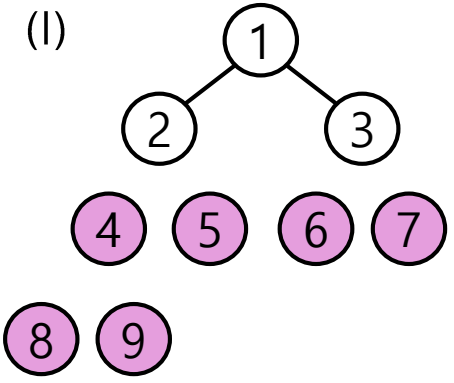
(j)


1
4
3
2
5
6
7
8
9

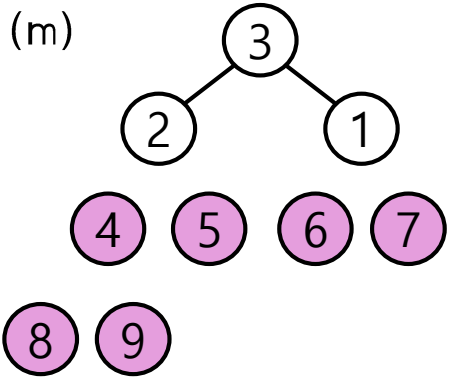
(k)


4
2
3
1
5
6
7
8
9

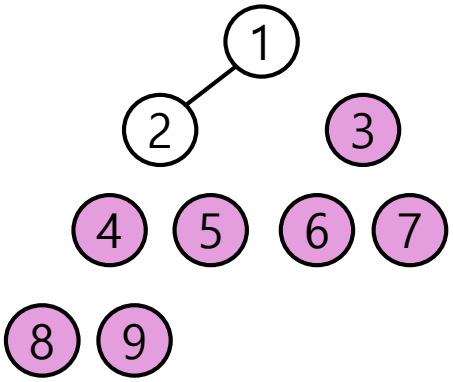
(l)


1
2
3
4
5
6
7
8
9

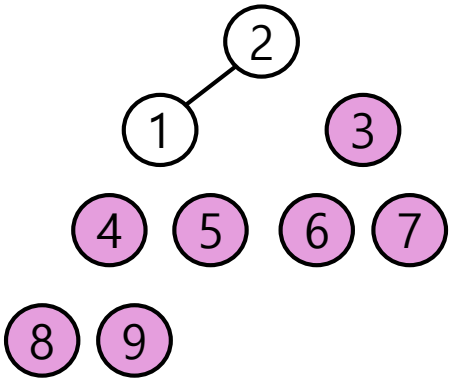
(m)


3
2
1
4
5
6
7
8
9

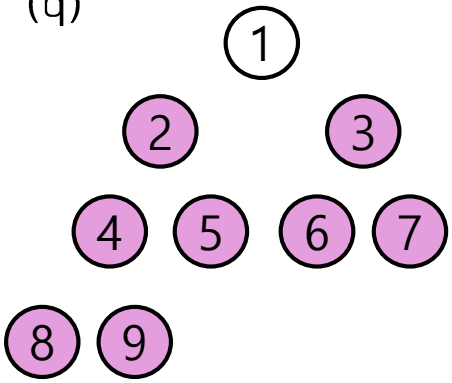
(m)



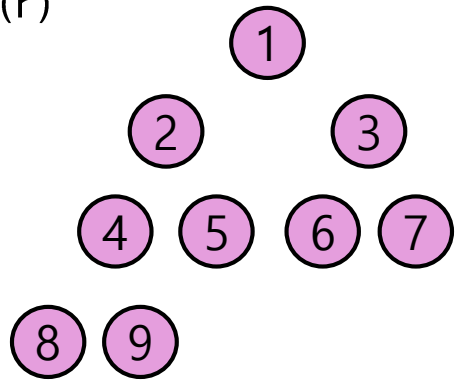
(o)



(q)



(r)



이 과정을 통하여 우리는 힙 정렬을 활용했다.

과정들을 정리하자면, 루트(Root)에 있는 값을 가장 뒤쪽으로 보내면서 힙 트리의 크기를 1씩 뺀 크기만큼 힙 생성 알고리즘을 수행하여 최대 힙을 유지해주고, 다시 조금 전 힙 생성 알고리즘을 수행했던 크기에서 동일한 과정을 반복했다.

전체 코드는 다음과 같다.

7-5. 힙 정렬

```python
def max_heapify(A, i, n):
    largest = i # 현재 노드를 가장 큰 값으로 가정
    left = 2 * i + 1 # 왼쪽 자식 노드의 인덱스
    right = 2 * i + 2 # 오른쪽 자식 노드의 인덱스

    # 왼쪽 자식 노드가 존재하고, 현재 노드보다 큰 경우
    if left < n and A[left] > A[largest]:
        largest = left

    # 오른쪽 자식 노드가 존재하고, 현재 노드보다 큰 경우
    if right < n and A[right] > A[largest]:
        largest = right

    # 가장 큰 값이 현재 노드가 아닌 경우
    if largest != i:
        # 현재 노드와 가장 큰 자식 노드를 교환
        A[i], A[largest] = A[largest], A[i]
        # 교환 후 영향을 받은 서브트리를 재귀적으로 힙화
        max_heapify(A, largest, n)

def build_max_heap(A):
    n = len(A)
    # 배열의 중간부터 시작하여 첫 번째 요소까지 반복
    for i in range(n // 2 - 1, -1, -1):
        max_heapify(A, i, n)

def heap_sort(A):
    n = len(A)
    build_max_heap(A) # 배열을 최대 힙으로 변환

# 하나씩 요소를 힙에서 추출하여 정렬
    for i in range(n - 1, 0, -1):
        A[0], A[i] = A[i], A[0] # 현재 루트(최댓값)와 마지막 요소를 교환
        max_heapify(A, 0, i) # 루트에서 힙 생성 알고리즘

# 사용 예시:
A = [9, 1, 8, 6, 4, 7, 2, 3, 5]
heap_sort(A)
print(A)
```

10-5. 출력 결과

```
[1, 2, 3, 4, 5, 6, 7, 8, 9]
```

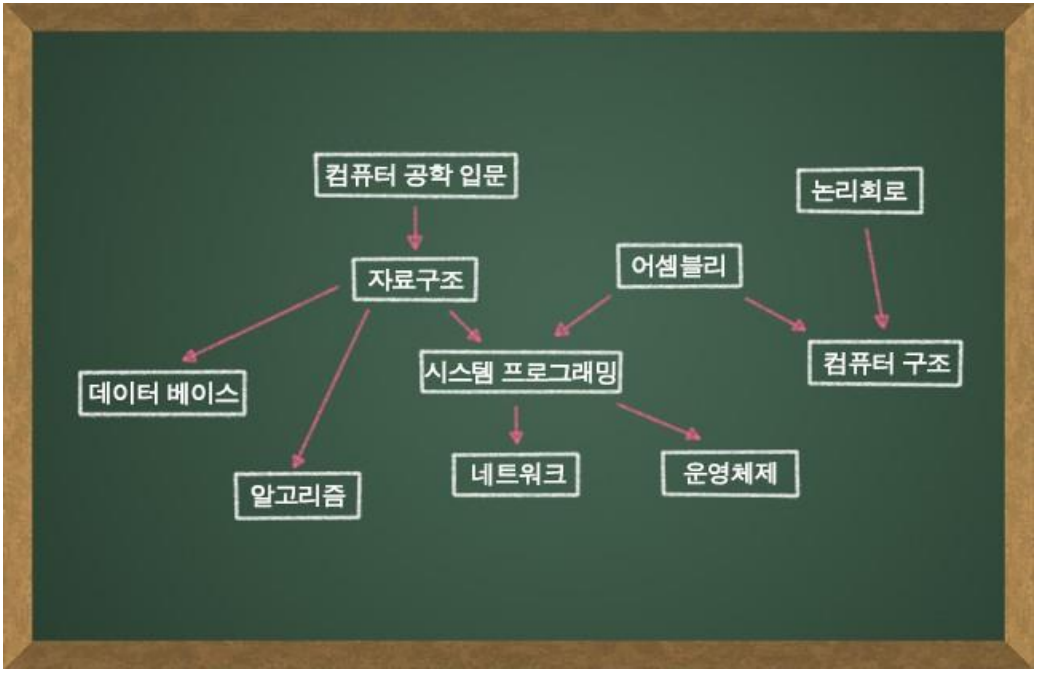
힙 정렬의 시간 복잡도는 O(N * log N) 이다. 사실상 모든 데이터를 기준으로 힙 생성 알고리즘을 쓰지 않아도 되기 때문에 O(N) 에 가까운 속도를 낼 수 있다.

힙 정렬은 병합 정렬과 다르게 별도로 추가적인 배열이 필요하지 않다는 점에서 메모리 측면에서 몹시 효율적이다. 또한 항상 O(N * log N)을 보장할 수 있다는 점에서 몹시 감력한 알고리즘이다. 그렇기에 이론적으로 퀵 정렬, 병합 정렬보다 더 우위에 있다고 할 수 있지만, 단순히 속도만 가지고 비교하면 퀵 정렬이 평균적으로 더 빠르기 때문에 힙 정렬이 일반적으로 많이 사용되지는 않는다.

03. 위상 정렬

위상 정렬

위상 정렬(Topology Sort)이란, “순서가 정해져 있는 작업”을 차례로 수행해야 할 때 그 순서를 결정해주기 위해 사용하는 알고리즘이다. 즉, 여러 가지 이들에 순서가 정해져 있을 때 순서에 맞게끔 나열하는 것을 의미한다.



위의 컴퓨터공학과 과목 순서도처럼, 일의 순서를 파악하여야 할 때, 위상 정렬을 사용하게 되면 일의 순서를 쉽게 알 수 있다.

위상 정렬을 알기 전에 이해를 도울 수 있는 관련 지식들부터 알아보도록 하자.

무방향 그래프

- 간선에 방향성이 주어지지 않은 그래프
- 두 정점 사이를 자유롭게 이동 가능

방향 그래프

- 간선에 방향성이 주어진 그래프
- 두 정점 사이 자유롭게 이동할 수 없고, 한쪽 방향으로만 이동 가능
- 자기 자신 노드로 이동 가능

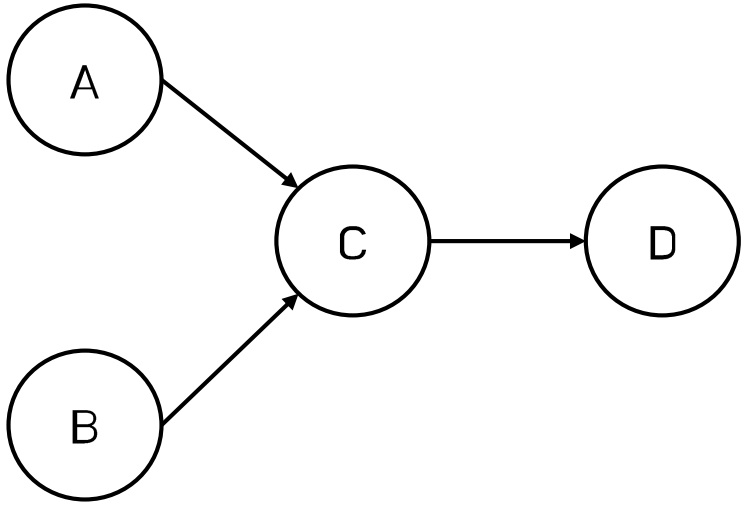
진입 차수

- 밖에서 한 정점으로 들어오는 간선의 수

진출 차수

- 한 정점에서 나가는 간선의 수

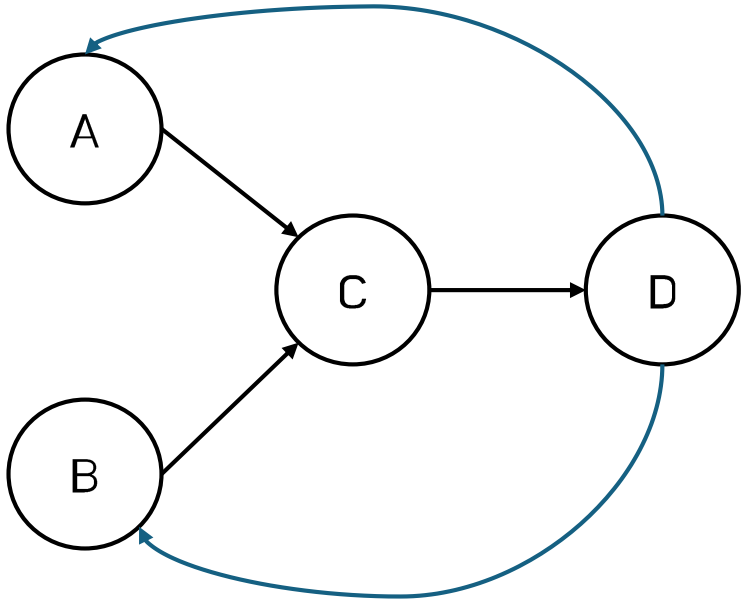
다음 방향 그래프에서 각 노드의 진입 차수와 진출 차수를 확인해 보도록 하자.



노드	A	B	C	D
진입 차수	0	0	2	1
진출 차수	1	1	1	0

진입 차수는 해당 노드에 들어오는 화살표의 개수, 진출 차수는 다른 노드로 나가는 화살표의 개수를 의미한다고 이해하면 좋다.

위상 정렬은 DAG(Directed Acyclic Graph)에서만 적용이 가능하다. 즉 사이클이 발생하지 않는 방향 그래프에서만 위상 정렬이 가능하다. 그러므로 사이클이 발생하는 경우 위상 정렬을 수행할 수 없다.



위와 같이 사이클이 발생하는 순간에 그래프에서 시작점을 찾을 수 없게 되고, 그로 인하여 순서를 정할 수 없게 된다.

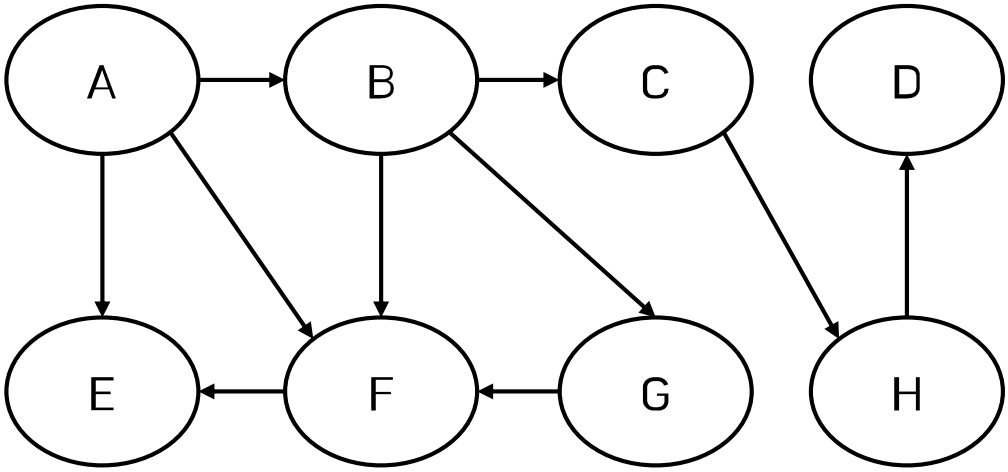
이를 토대로 위상 정렬은 두 가지 해결책을 낼 수 있다.

- 현재 그래프는 위상 정렬이 가능한가?
- 위상 정렬이 가능하다면 그 결과는 무엇인가?

사이클이 존재한다면 위상 정렬이 불가능하다 라는 결론을 얻을 수 있고, 사이클이 존재하지 않은 방향 그래프(DAG)라면 순서를 파악할 수 있다.

위상 정렬은 스택/큐 둘 다 구현이 가능하나, 큐를 이용하는 방법을 소개하도록 하겠다.

(1)진입 차수, 진출 차수 정리하기



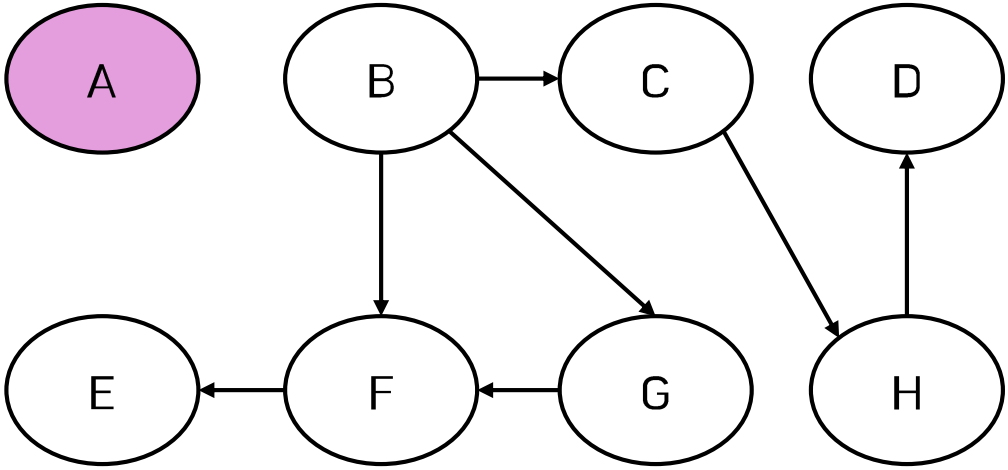
위 그래프의 진입차수와 진출 차수를 정리한다.

노드	A	B	C	D	E	F	G	H
진입	0	1	1	1	2	3	1	1
진출	3	3	1	0	0	1	1	1

(2) 진입 차수가 0인 정점을 큐에 삽입

현재 진입 차수가 0인 노드는 A밖에 존재하지 않으니, 큐에 추가해준다.

(3) 큐에서 원소를 꺼내 연결된 모든 간선을 제거하기



큐에 있는 A를 꺼냈기에, 큐는 현재 비어있는 상태이다.

(4) 간선 제거 후 진입/진출 차수 정리하기

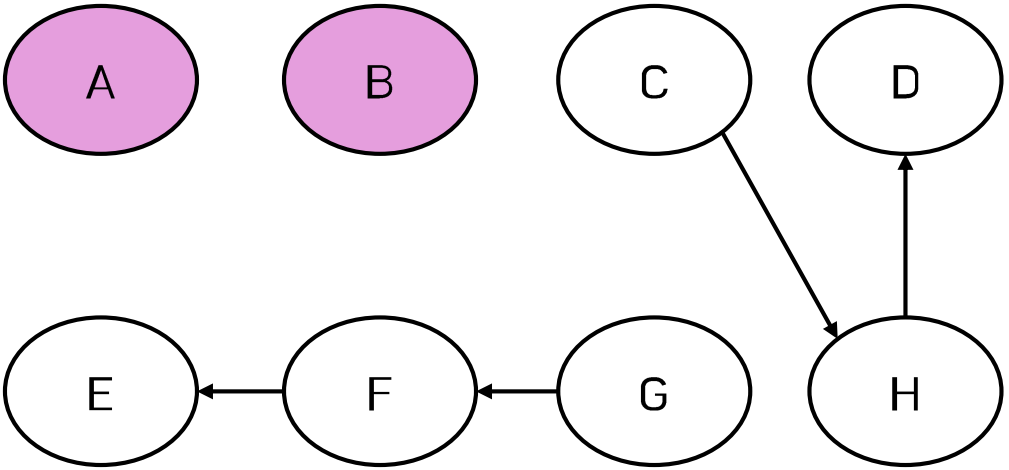
노드	A	B	C	D	E	F	G	H
진입	0	0	1	1	1	2	1	1
진출	0	3	1	0	0	1	1	1

A는 제거가 된 나머지 노드들의 진입, 진출 차수가 재정리 되었다.

(4) 진입 차수가 0이 된 정점을 큐에 삽입하기

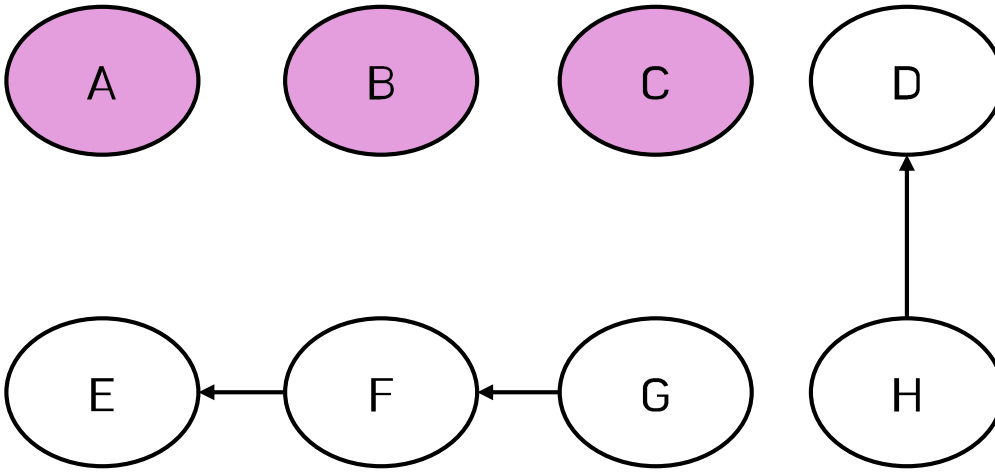
진입차수가 0인 정점은 현재 B노드밖에 없기에, 큐에 추가한다.

(5) 2~4번 과정을 반복한다.



노드	A	B	C	D	E	F	G	H
진입	0	0	0	1	1	1	0	1
진출	0	0	1	0	0	1	1	1

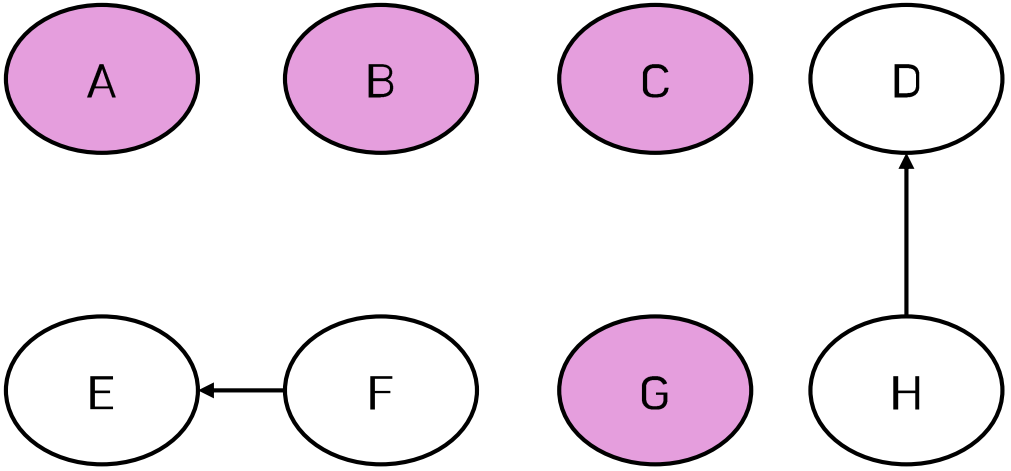
C, G가 큐에 추가되었고, 맨 앞에 있는 C를 pop 한다.



노드	A	B	C	D	E	F	G	H
진입	0	0	0	1	1	1	0	0
진출	0	0	0	0	0	1	1	1

이 후, 진입/진출 차수를 구하고, 진입 차수가 0인 H를 큐에 추가한다.

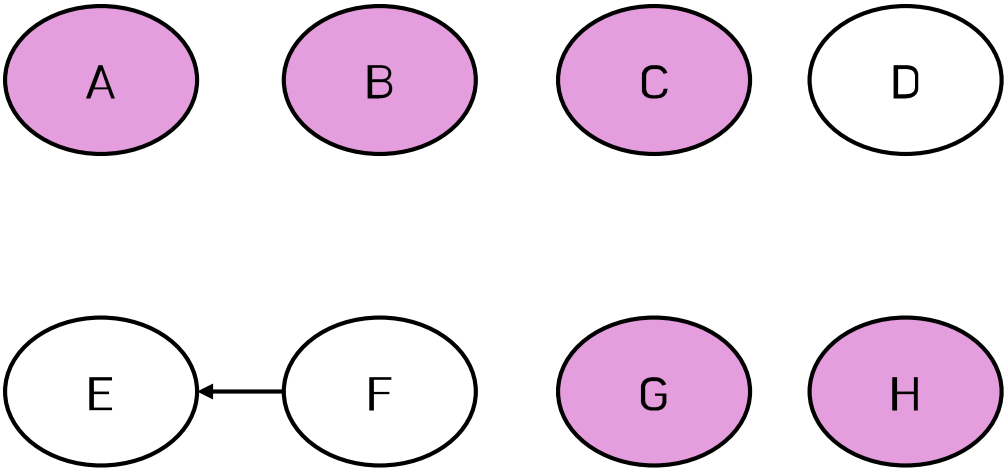
큐의 맨 앞인 G를 pop한 후에 마저 반복한다.



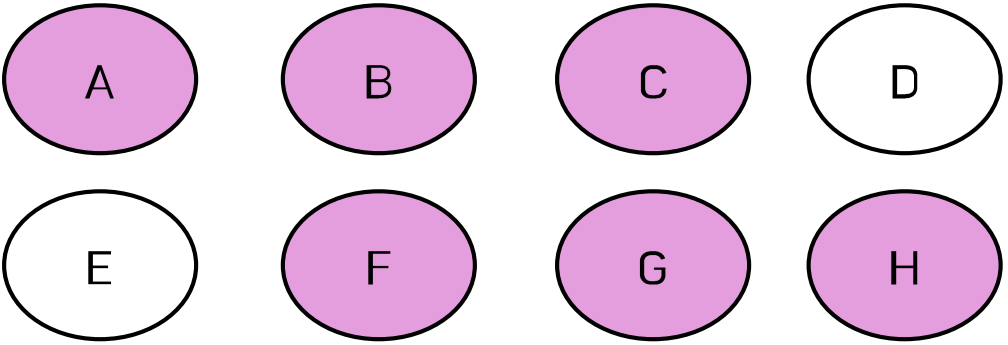
노드	A	B	C	D	E	F	G	H
진입	0	0	0	1	1	0	0	0
진출	0	0	0	0	0	1	1	1

진입 차수가 0인 F를 큐에 추가한다.

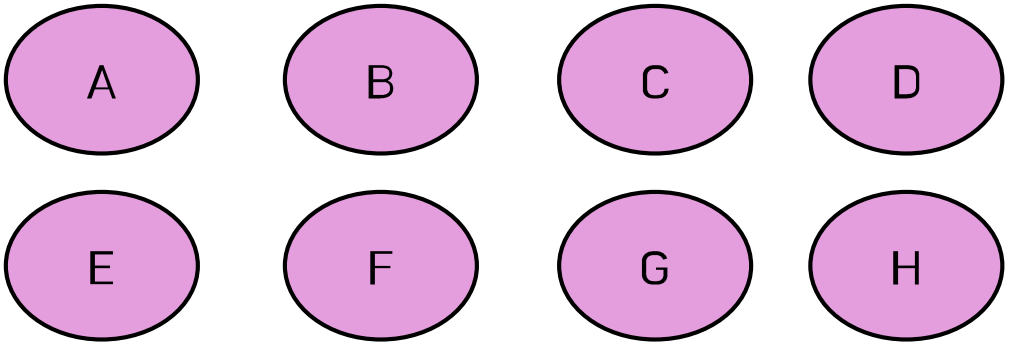
그 후, 큐에서 맨 앞인 H를 pop 한 후에 마저 반복한다.



노드	A	B	C	D	E	F	G	H
진입	0	0	0	0	1	0	0	0
진출	0	0	0	0	0	1	1	0



노드	A	B	C	D	E	F	G	H
진입	0	0	0	0	0	0	0	0
진출	0	0	0	0	0	0	0	0



이렇게 해서 순서는 A → B → C → G → H → F → D → E가 위상 정렬의 결과로 나오게 되었다.

만약 2~4번까지의 과정을 반복하는 중간에, 4번 이후 모든 노드를 아직 방문하지 않았으나 큐가 비어있다면 사이클이 존재하는 것이기에 위상 정렬의 결과가 나올 수 없다. 모든 원소를 방문 하였다면, 큐에서 꺼낸 순서가 위상 정렬의 결과이다.

위상 정렬의 코드는 다음과 같다.

7-6. 위상 정렬(1)

```python
from collections import deque

def topological_sort(vertices, edges):
    # 그래프 초기화
    graph = {vertex: [] for vertex in vertices}
    in_degree = {vertex: 0 for vertex in vertices}

    # 간선 추가 및 진입 차수 계산
    for u, v in edges:
        graph[u].append(v)
        in_degree[v] += 1

        # 진입 차수가 0인 노드들을 큐에 추가
        queue = deque([v for v in vertices if in_degree[v] == 0])
        top_order = []

    # 위상 정렬 수행
    while queue:
        u = queue.popleft()
        top_order.append(u)

        # 해당 노드의 인접 노드들의 진입 차수 감소
        for v in graph[u]:
            in_degree[v] -= 1
            # 진입 차수가 0이 된 노드들을 큐에 추가
            if in_degree[v] == 0:
                queue.append(v)

    # 사이클이 있는 경우 (모든 노드가 위상 정렬에 포함되지 않은 경우)
    if len(top_order) != len(vertices):
        raise ValueError("그래프에 사이클이 존재합니다!")

    return top_order
```

7-6. 위상 정렬(2)

```
# 사용 예시:
vertices = ["A", "B", "C", "D", "E", "F", "G", "H"]
edges = [("A", "B"), ("A", "E"), ("A", "F"), ("B", "C"), ("B", "F"),
("B", "G"), ("C", "H"), ("F", "E"), ("G", "F"), ("H", "D")]

try:
order = topological_sort(vertices, edges)
print("위상 정렬 순서:", order)
except ValueError as e:
print(e)
```

7-6. 출력 결과

위장 성렬 순서 : ['A', 'B', 'C', 'G', 'H', 'F', 'D', 'E']

아까 위상 정렬의 과정에서 나온 값과 같은 결과가 나왔다.

다만, 동일한 순서에 있는 노드의 경우 순서가 바뀔 수도 있다.

제 8 장

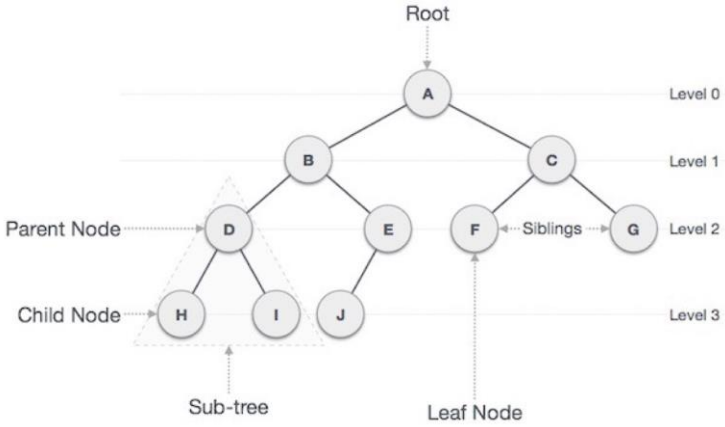
트리

1) 트리란?
2) 트리의 종류
3) 트리 순회

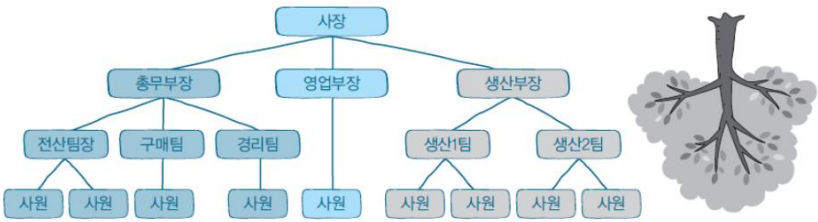
01. 트리란?

트리의 정의

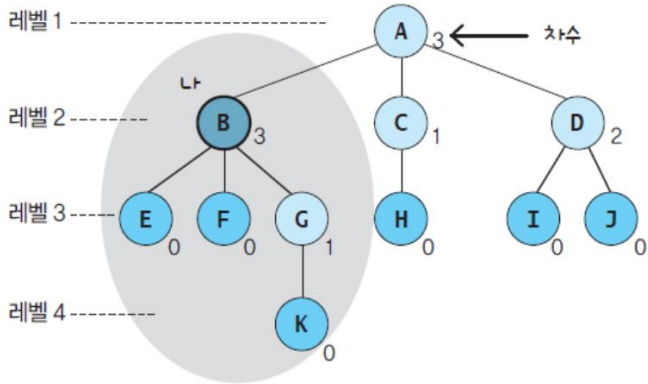
트리(tree)란 어떤 노드들의 집합으로 노드들은 각각 서로 다른 자식을 가지면서 이때의 각 노드들은 재사용 되지 않는 구조이다.



트리는 나무 모양의 자료구조로써, 계층적인 관계를 가진 자료의 표현에 매우 유용하며, 회사의 조직도, 나무의 잎과 같은 형태의 비선형 자료구조에 해당한다.



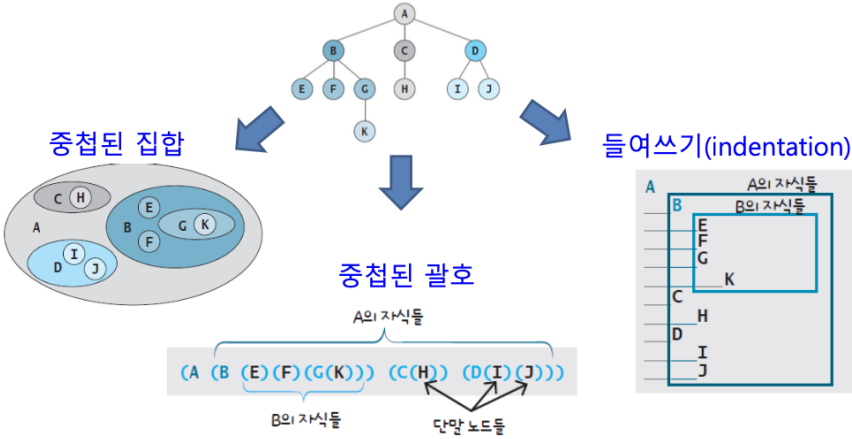
트리 관련 용어



- 노드 : 값과 부모의 자식 정보를 가지고 있음
- 엣지/간선 : 부모 노드와 자식 노드를 연결
- 루트 노드 : A
- B의 부모노드 : A
- B의 자식 노드 : E, F, G
- B의 자손 노드 : G, B, A
- K의 조상 노드 : C, D
- B의 차수 : 3
- 단말 노드 : E, F, K, H, I ,J
- 비단말 노드 : A, B, K, H, I, J
- 트리의 높이: 4
- 트리의 차수: 3

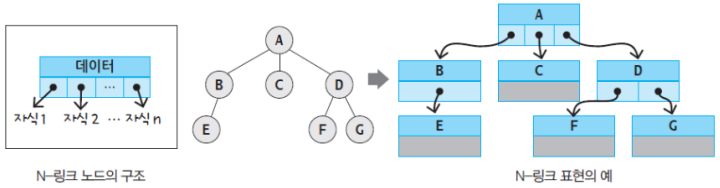
트리의 표현 방법

- 노드와 간선의 연결 관계

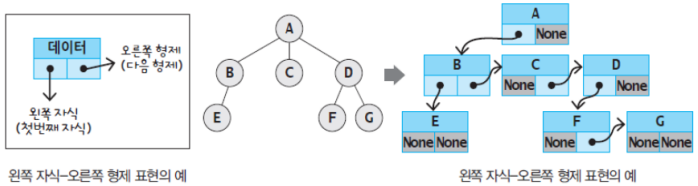


일반 트리의 표현

- 자식의 개수에 제한이 없는 트리(general tree)
- 방법1: N 링크 표현



- 방법2: 왼쪽 자식-오른쪽 형제 표현

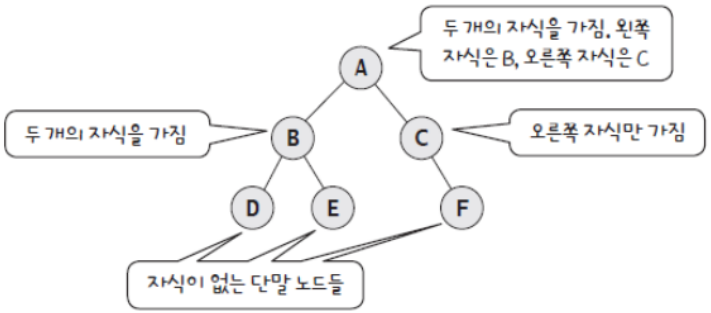


02. 트리의 종류

이진 트리란?

모든 노드가 최대 2개의 자식만을 가질 수 있는 트리를 말한다.

- 모든 노드의 차수가 2 이하로 제한
- 왼쪽 자식과 오른쪽 자식은 반드시 구별

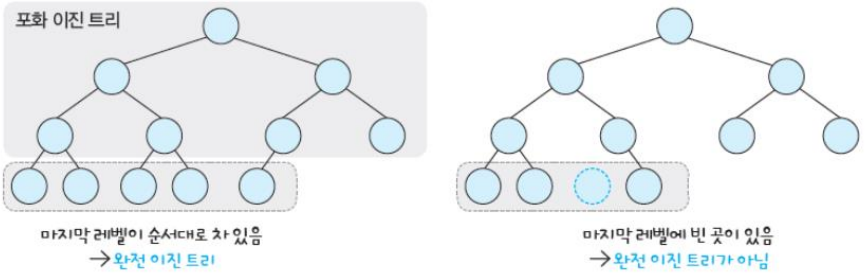


이진 트리의 예시

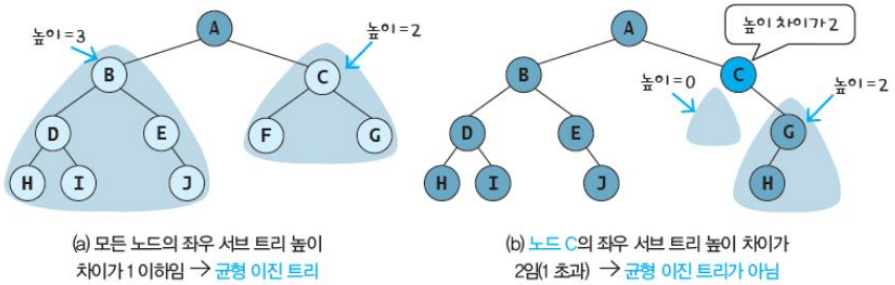
- 빠른 자료의 탐색이 가능한 이진 탐색 트리(binary search tree)
- 우선순위 큐를 효과적으로 구현하는 힙 트리(heap tree)
- 수식을 트리 형태로 표현하여 계산하는 수식 트리 등

트리의 종류

- 완전 이진 트리(complete binary tree)
  - 레벨 k-1까지는 포화 이진트리
  - 마지막 레벨 k에서는 왼쪽부터 오른쪽으로 노드가 순서대로

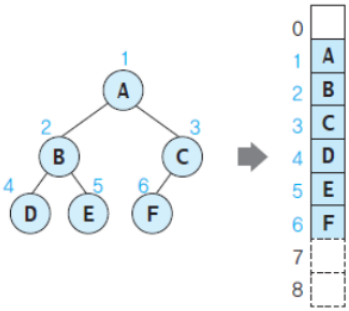


- 균형 이진 트리(balanced binary tree)
  - 높이 균형 이진 트리(height-balanced binary tree)
  - 모든 노드에서 좌우 서브 트리의 높이 차이가 1 이하인 트리
  - 그게 아니라면 → 경사 트리

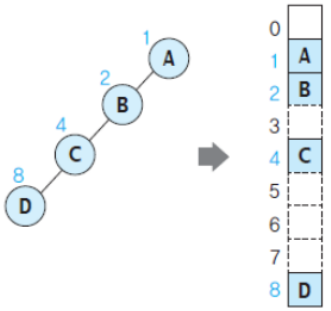


이진 트리의 배열 구조 표현

- 배열 구조
  - 이진 트리를 포화 이진 트리의 일부라고 생각하고 번호 부여



(a) 완전 이진 트리의 배열 표현. 중간에 빈 칸이 발생하지 않음.



(b) 경사 이진 트리의 배열 표현. 중간에 빈 칸이 많이 발생할 수 있음.

- 노드 i의 부모 노드의 인덱스 = i/2
- 노드 i의 왼쪽 자식 노드 인덱스 = 2i
- 노드 i의 오른쪽 자식 노드 인덱스 = 2i + 1

파이썬 연산자 나눗셈은 /와 // 로 구분되어 있다.

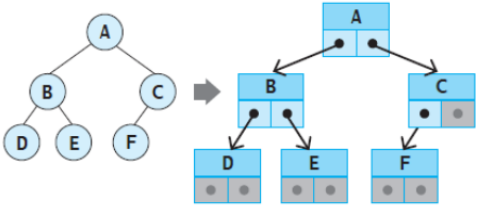
코드에서 사용할 때에는 정수 나눗셈을 위한 //를 사용하여 i//2라 표현해야 한다.

이진 트리의 링크 표현법

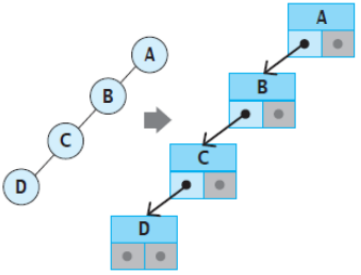
- 연결된 구조 표현: 링크 표현법
  - 연결된 구조의 이진 트리를 위한 노드의 구조
  - 데이터안에 왼쪽자식(left), 오른쪽자식(right)의 구조를 가진 노드

8-1. 이진트리를 위한 노드의 생성자

```
1  Class BTNode:
2          def __init__(self, elem, left=None, right=None):
3                  self.data = elem
4                  self.left = left #왼쪽 자식을 위한 링크
5                  self.right = right #오른쪽 자식을 위한 링크
```



(a) 완전 이진 트리의 링크 표현

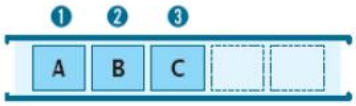


(b) 경사 이진 트리의 링크 표현

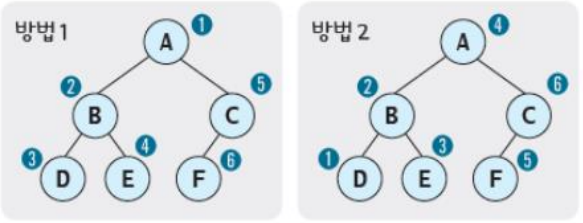
03. 트리 순회

순회란(traversal)?

트리의 모든 노드를 한번씩 방문 하는 것을 의미한다.



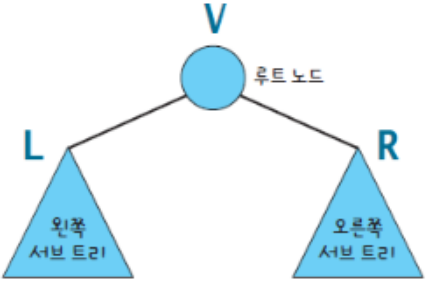
선형자료구조는
순회 방법이 단순합니다.



트리는 다양한 방법으로 순회할 수 있습니다.

이진 트리의 표준 순회 3가지

- 전위 순회(preorder traversal) : VLR
- 중위 순회(inorder traversal) : LVR
- 후위 순회(postorder traversal) : LRV



전위 순회(preorder)

- V → L → R (서브 트리도 같은 순회 방법을 적용)
  - 연결된 구조의 이진 트리를 위한 노드의 구조
  - 데이터안에 왼쪽자식(left), 오른쪽자식(right)의 구조를 가진 노드

8-2. 전위 순회

```
1  def preorder(n):
2          if n is not None:
3                  print(n.data, end=' ') #노드를 방문해 처리할 연산들의 위치
4                  preorder(n.left) #왼쪽 서브 트리 처리
5                  preorder(n.right) #오른쪽 서브 트리 처리
```

8-2. 출력 결과

```
A B D E C F
```

중첩된 괄호 표현 : (A (B (D) (E)) (C (F)))

중위 순회(inorder)

- L → V → R

8-3. 전위 순회

```
1  def inorder(n):
2          if n is not None:
3                  inorder(n.left) #왼쪽 서브 트리 처리
4                  print(n.data, end=' ') #노드에서 처리할 연산들의 위치
5                  inorder(n.right) #오른쪽 서브 트리 처리
```

8-3. 출력 결과

```
D B E A C F
```

후위 순회(postorder)

- L → R → V

8-4. 후위 순회

```
1  def postorder(n):
2          if n is not None:
3                  postorder(n.left) #왼쪽 서브 트리 처리
4                  postorder(n.right) #오른쪽 서브 트리 처리
5                  print(n.data, end=' ') #노드에서 처리할 연산들의 위치
```

8-3. 출력 결과

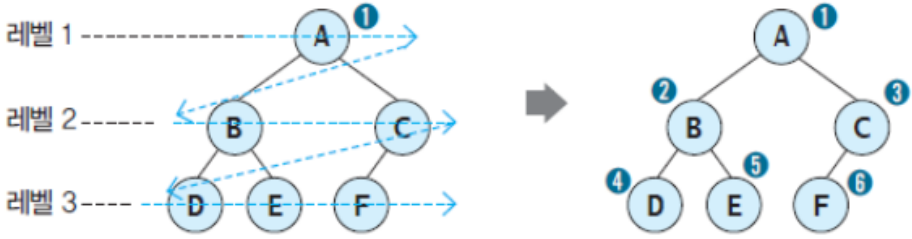
```
D E B F C A
```

순회 방법의 선택

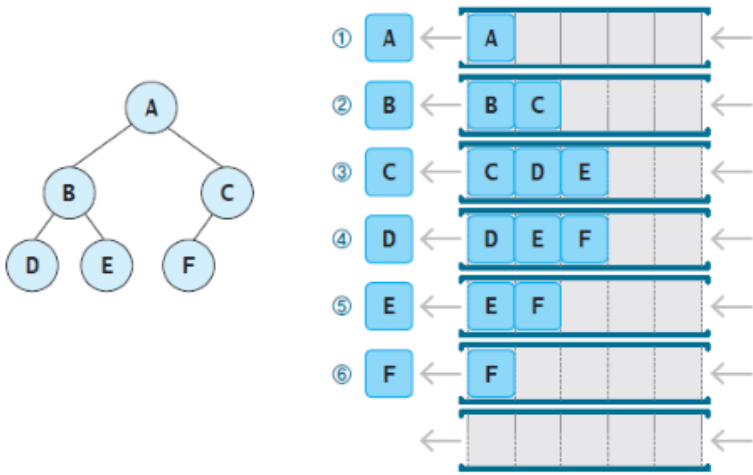
- 순회 방법은 어떻게 선택할까 ?
    - 순서는 중요하지 않고 노드를 전부 방문하기만 하면 될 때 ?
        - 어떤 방식을 사용해도 상관 없다!
        - (예) 노드의 수, 모든 노드의 순서 없는 출력 등
    - 자식을 먼저 처리해야 부모를 처리할 수 있을 때 ?
        - 후위 순회를 선택하면 된다!
        - (예) 컴퓨터 폴더의 용량 계산
    - 부모가 처리되어야 자식을 처리할 수 있을 때 ?
        - 전위 순회를 선택하면 된다!
        - (예) 노드의 레벨 계산

레벨 순회(level order)

- 레벨 순으로 노드를 방문

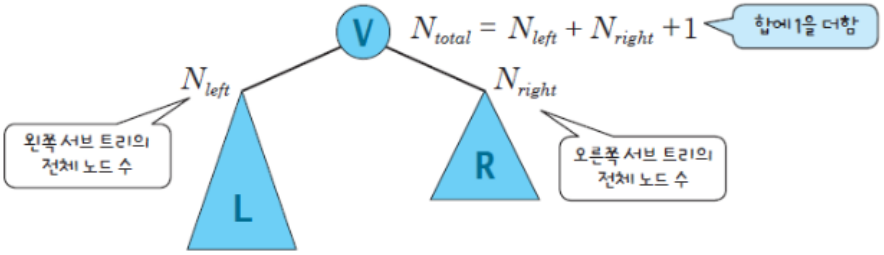


- 큐로 구현



이진 트리의 연산들

- 전체 노드의 수 구하기
    - 이진 트리의 노드 개수는 왼쪽 서브 트리의 노드 수와 오른쪽 서브 트리의 노드 수의 합에 1(루트 노드)을 더하면 된다.

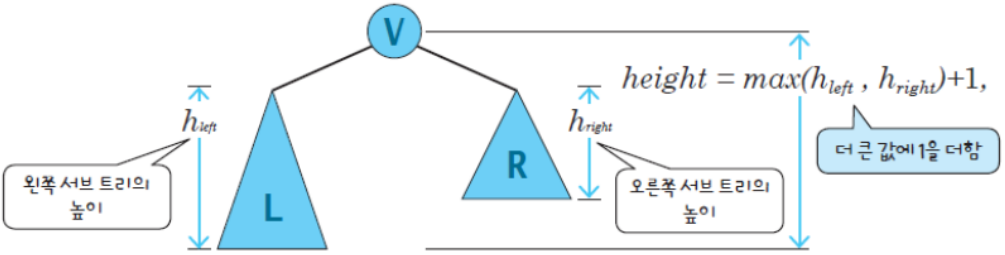


8-5. 전체 노드의 수 구하기

```
1  def count_node(n):
2          if n is None:  #n이 None이면 공백 트리 -> 0 반환
3                  retrun 0
4          else:          #좌우 서브 트리의 노드 수의 합 + 1 (순환이용)
5                  return count_node(n.left) + count_node(n.right) + 1
```

트리의 높이 구하기

- 이진 트리의 높이는 왼쪽 서브 트리의 높이와 오른쪽 서브 트리의 높이 중에서 큰 값에 1을 더한 값 이다.



8-6. 트리의 높이 구하기

```
1  def calc_height(n):
2          if n is None:  #n이 None이면 공백 트리 -> 0 반환
3                  retrun 0
4          hLeft = calc_height(n.left) #hLeft <- L의 높이
5          hRight = calc_height(n.right) #hRight <- R의 높이
           if(hLeft > hRihgt) : #더 큰 값에 1을 더해 반환
                  return hLeft + 1
           else: return hRight + 1
```

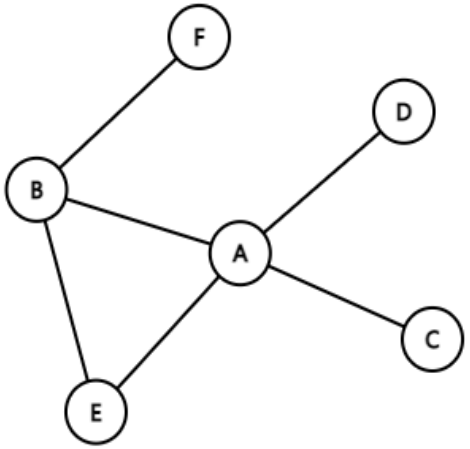
제 9 장

그래프

1) 그래프란?
2) 그래프 순회 알고리즘
3) 최단 경로 알고리즘

01. 그래프란?

그래프의 정의



그래프란 정점(Vertex/Node)와 간선(edge)로 구성된 비선형 자료구조이다.

그래프 G=(V,E)로 표현하는데 V는 정점의 집합이고 E는 간선의 집합을 의미한다.

그래프와 트리의 차이

- 트리는 그래프의 한 종류이다.
- 트리는 방향을 갖지만 그래프는 방향을 가질 수도, 가지지 않을 수도 있다.
- 트리는 한 개의 루트 노드를 갖지만 그래프는 루트 노드 개념이 없다.
- 그래프는 부모-자식 개념이 없다.

그래프를 이해하기 위한 관련 용어는 다음과 같다.

인접 정점

- 간선에 의해 직접 연결되는 정점

차수

- 무방향 그래프에서 정점에 연결된 인접 정점의 개수

진출/진입 차수

- 방향 그래프에서 외부로 향하는/외부에서 들어오는 간선의 수

단순 경로

- 경로 중에서 반복되는 경로가 없는 경우

사이클

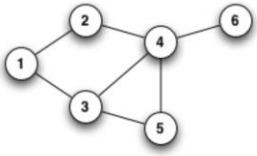
- 단순 경로에서 시작 정점과 종료 정점이 같은 경우

경로 길이

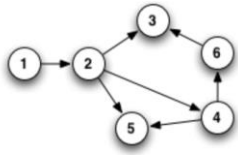
- 경로를 구성하는데 사용된 간선의 수를 의미

그래프의 종류

무방향 그래프 VS 방향 그래프



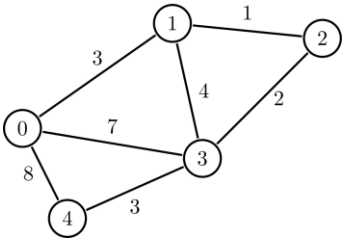
Undirected Graph



Directed Graph

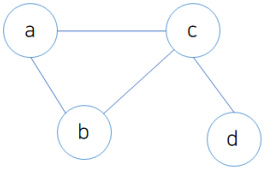
그래프는 방향이 존재할 수도, 존재하지 않을 수도 있다.

가중치 그래프 VS 비가중치 그래프

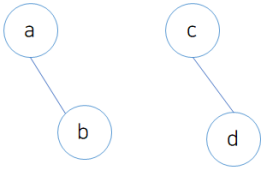


그래프의 간선은 가중치를 가질 수도, 갖지 않을 수도 있다.

연결 그래프 VS 비연결 그래프



연결 그래프



비연결 그래프

그래프의 모든 노드는 직,간접적으로 연결 될 수도, 연결되지 않을 수도 있다.

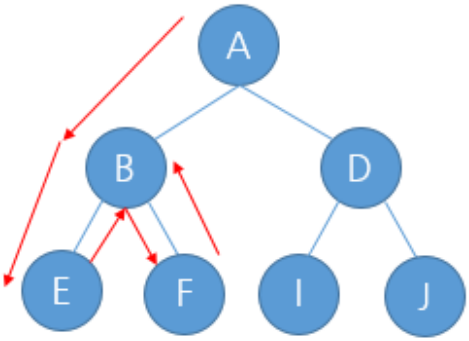
02. 그래프 순회 알고리즘

그래프 탐색이란?

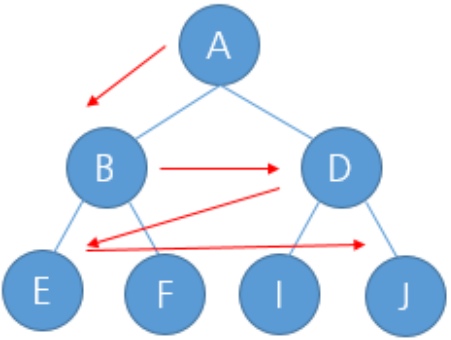
하나의 정점으로부터 시작하여 차례대로 모든 정점들을 한 번씩 방문하는 것이 그래프 탐색이다. 소셜 네트워크에서 사용자들 간의 관계를 분석하거나, 네비게이션에서 최단 경로를 탐색할 때 사용될 수 있다.

그래프 탐색 알고리즘은 **깊이 우선 탐색**(DFS, Depth-First Search) 알고리즘과 **너비 우선 탐색**(BFS, Breadth-First Search) 알고리즘으로 나뉜다.

DFS(깊이우선탐색)



DFS
A B E F D I J 순서로 방문

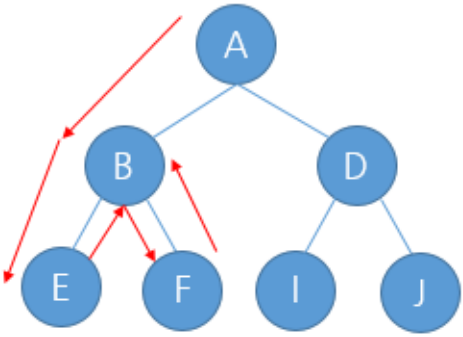


BFS
A B D E F I J 순서로 방문

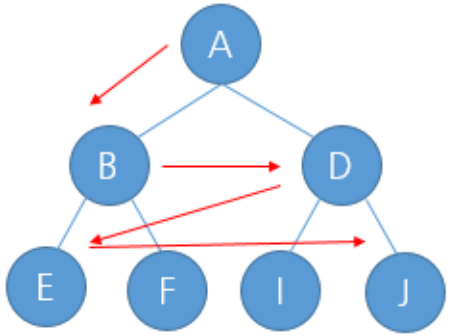
DFS는 루트 노드(또는 임의의 노드)에서 시작해 다음 분기로 넘어가기 전에 해당 분기를 완벽하게 탐색하는 방법이다.
탐색 시 한 방향으로 갈 수 있을 때까지 탐색하며 나아가다 더이상 갈 수 없게 되면 가장 최근의 갈림길로 돌아와서 다른 방향으로 탐색을 진행한다. 이를 백트래킹(back-tracking)이라 부르며, DFS의 핵심 개념이다.

DFS는 재귀를 사용하기 때문에 일반적으로 BFS보다 속도는 느리다. 단 모든 노드를 방문해야 하는 문제에서 효과적으로 사용될 수 있다.
DFS 알고리즘을 구현할 때 유의할 점은, 무한 재귀에 빠지지 않도록 방문 여부를 설정해야 한다는 점이다.

BFS(너비우선탐색)



DFS
A B E F D I J 순서로 방문



BFS
A B D E F I J 순서로 방문

BFS(Breadth-First Search)는 시작 노드에서 가까운 노드부터 먼 노드 순으로 탐색하는 방법이다. BFS는 큐(Queue)를 사용하여 탐색을 진행하며, FIFO(First-In-First-Out) 원칙을 따른다. 시작 노드에서 출발해 인접한 노드를 모두 탐색한 후, 인접한 노드의 인접 노드를 탐색하는 방식이다.

BFS는 최단 경로 탐색에 효과적이며, 재귀를 사용하진 않지만 무한 루프에 빠질 수 있으므로 DFS와 마찬가지로 방문 여부를 설정해야 한다.

DFS(깊이우선탐색) 코드

9-1. DFS 코드

```
1  def dfs(graph, node, visited=set()):
2   if node not in visited:
3    print(node)
4    visited.add(node)
5    for neighbor in graph[node]:
6     dfs(graph, neighbor, visited)
7
8  graph = {'A': ['B', 'C'], 'B': ['D', 'E'], 'C': ['F'], 'D': [], 'E': ['F'], 'F': []}
9  dfs(graph, 'A')
```

9-1. 출력 결과

```
A
B
D
E
F
C
```

2~4행 방문되지 않은 노드라면 출력하고 방문 처리한다.

5~6행 그래프를 반복문으로 돌며 재귀로 그래프를 방문한다.

9행 함수를 처음 호출한다.

이렇듯 DFS에선 방문 처리와 재귀가 핵심 요소인 것을 알 수 있다.

BFS(너비우선탐색) 코드

9-2. BFS 코드

```
1  from collections import deque
2  def bfs(graph, start):
3   visited = set()
4   queue = deque([start])
5   while queue:
6    node = queue.popleft()
7    if node not in visited:
8     print(node)
9     visited.add(node)
10    for neighbor in graph[node]:
11     if neighbor not in visited:
12      queue.append(neighbor)
13
14 graph = {'A': ['B', 'C'], 'B': ['D', 'E'], 'C': ['F'], 'D': [], 'E': ['F'], 'F': []}
15 bfs(graph, 'A')
```

9-2. 출력 결과

```
A
B
C
D
E
F
```

5행

덱이 비지 않을 때까지 반복문을 수행한다.

7~9행

방문되지 않은 노드라면 출력하고 방문 처리한다.

10~12행

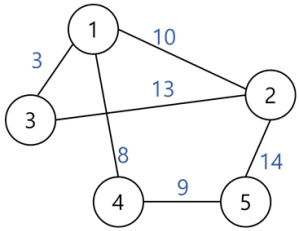
그래프를 돌며 방문하지 않은 노드라면 덱에 넣는다.

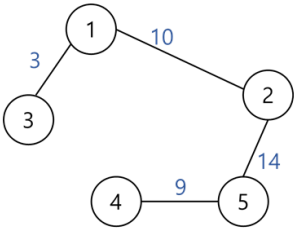
DFS의 핵심 요소가 재귀였다면 BFS의 핵심 요소는 큐를 반복문으로 도는 것이다. 방문 여부를 확인하는 것은 같다.

Queue 대신 Deque를 쓰는 이유는 Queue의 pop(idx)보다 Deque의 popleft()가 시간 복잡도면에서 더 우수하기 때문에 알고리즘 문제를 풀 때 더 유리하기 때문이다.

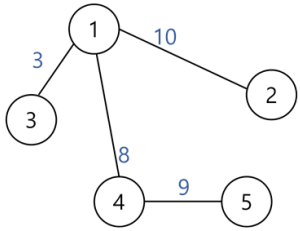
03. 최단 경로 알고리즘

최소신장트리란?

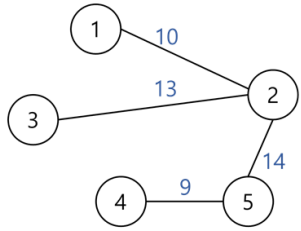




가중치 합: 36



가중치 합: 30



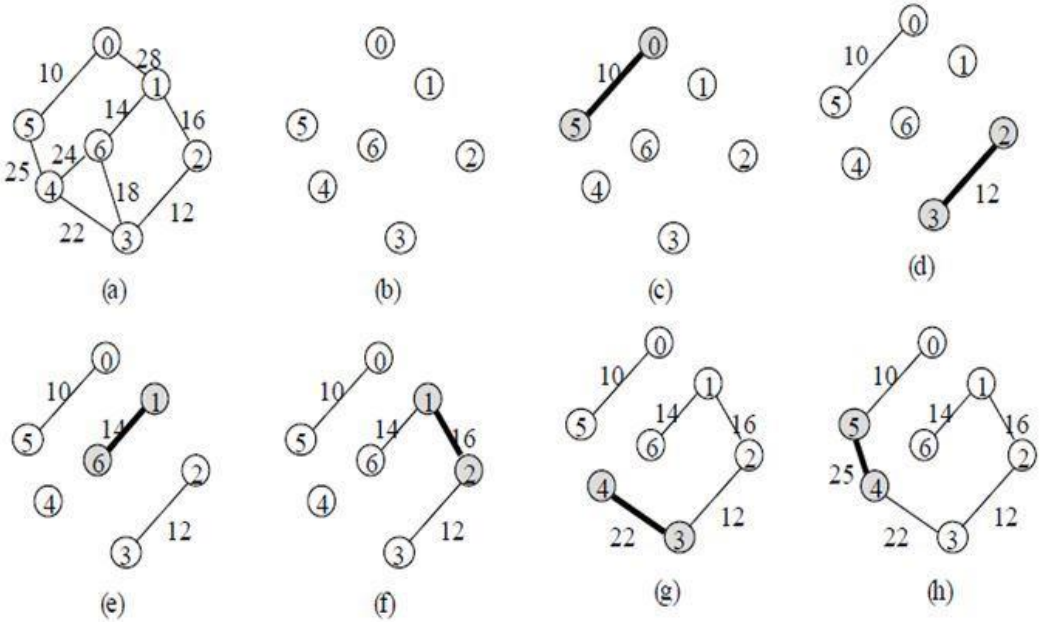
가중치 합: 46

최소신장트리는 가중치 그래프에서 사이클 없이 최소 가중치로 모든 노드를 연결하는 자료구조이다.

통신 네트워크로 생각해 보자. 모든 기지국을 최소한의 설치 비용으로 연결하려 할 때 최소신장트리 알고리즘을 사용해 설계할 수 있을 것이다.

최소신장트리를 찾는 알고리즘은 Kruskal 알고리즘과 Prim 알고리즘이 있다.

Kruskal 알고리즘



Kruskal 알고리즘의 각 단계

크루스칼의 핵심은 각 단계에서 **가중치가 가장 작은 간선부터 선택**하되, 사이클이 형성될 경우 해당 간선은 포함하지 않는 것으로, 그리디 알고리즘에 해당한다.

- 가장 작은 가중치를 선택하기 위해 매 단계마다 가중치를 정렬한다.
- 사이클 형성 여부를 확인하기 위해 Unoin-Find 구조를 사용한다.

Kruskal 알고리즘 코드

9-3. 크루스칼 코드

```
1  import sys
2  input=sys.stdin.readline
3
4  def find(x):
5   if parent[x] != x:
6    parent[x]=find(parent[x])
7   return parent[x]
8
9  def union(x, y):
10  a, b = find(x), find(y)
11  if a != b:
12   parent[a] = b
13
14 V, E = map(int, input().split())
15 arr, parent = [], [i for i in range(V)]
16
17 for _ in range(E):
18  v1, v2, dist = map(int, input().split())
19  arr.append([dist, v1-1, v2-1])
20
21 arr.sort()
22 mst_cost = 0
23
24 for a in arr: # 가중치를 차례대로 선택할거다
25  if find(a[1]) == find(a[2]): # 부모가 같다 = 같은 집합이다 = 사이클
26   continue # 포함하지 않는다
27
28  union(a[1], a[2]) # 부모가 다르다면 트리 포함
29  mst_cost += a[0] # 가중치 누적
30
31 print(mst_cost )
```

9-3. 입력

```
4 5
1 2 1
2 3 2
3 4 3
4 1 4
1 3 5
```

9-3. 출력 결과

```
6
```

4행

find 메소드, 재귀를 타고 올라가며 요소의 최종 부모를 찾는다.

9행

union 메소드, 서로 다른 부모를 가진 집합을 통합한다.

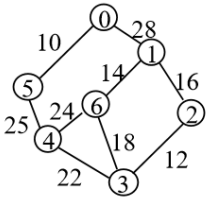
24행

최소신장트리는 사이클은 포함하지 않으므로 사이클 형성 시 건너뛴다.

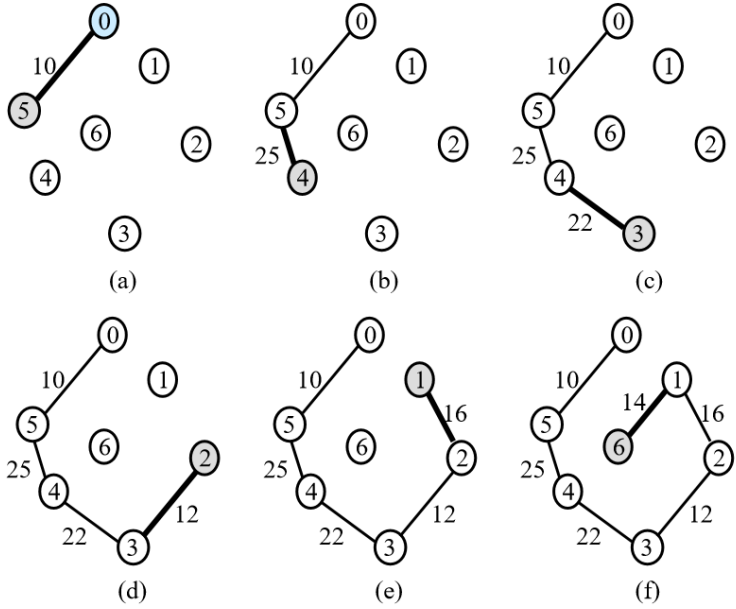
28행

부모가 다르다면 사이클이 아니므로 트리에 포함시킨다.

Prim 알고리즘



주어진 그래프



Prim 알고리즘의 진행 단계

프림과 크루스칼의 가장 큰 차이점은 크루스칼은 모든 정점에 대해 최소 가중치 간선을 선택해 나가지만 프림은 트리에 연결된 간선 중 최소 가중치 간선을 포함시켜 나간다는 점이다.

- 시작 단계: 시작 정점만 포함
- 인접 정점 중 최소 간선으로 연결된 정점 선택
- 우선순위 큐를 사용

프림 알고리즘 코드

9-4. 프림 코드

```
import sys
import heapq
input = sys.stdin.readline

def prim(start, graph, V):
    mst_cost = 0
    visited = [False] * V
    min_heap = [(0, start)]
    edges_count = 0

    while min_heap and edges_count < V:
        weight, current_vertex = heapq.heappop(min_heap)
        if visited[current_vertex]:
            continue

        visited[current_vertex] = True
        mst_cost += weight
        edges_count += 1

        for neighbor, edge_weight in graph[current_vertex]:
            if not visited[neighbor]:
                heapq.heappush(min_heap, (edge_weight, neighbor))

    return mst_cost

V, E = map(int, input().split())
graph = [[] for _ in range(V)]

for _ in range(E):
    v1, v2, dist = map(int, input().split())
    graph[v1-1].append((v2-1, dist))
    graph[v2-1].append((v1-1, dist))

result = prim(0, graph, V)
print(result)
```

9-4. 입력

```
4 5
1 2 1
2 3 2
3 4 3
4 1 4
1 3 5
```

9-4. 출력 결과

```
6
```

8행
(가중치, 정점) 형태의 우선순위 큐

11행~22행

힙이 비어있지 않고 mst의 간선의 수가 V보다 작을 때까지 while문을 실행한다.

만약 정점이 방문되었다면 건너뛰고 방문되지 않았다면 mst에 포함시키며

가중치를 합산하고 간선의 수를 증가시킨다.

제 10 장

연습문제 및 해설

1) 연습문제
2) 해설

연습문제 01.

문제

N개의 정수가 주어졌을 때 홀수인지 짝수인지 판별하는 프로그램을 작성하시오.

입력

첫째 줄에 숫자의 개수 $N(1 \leq N \leq 100)$이 주어진다.

둘째 줄부터 $N+1$개의 줄에 걸쳐 홀수, 짝수를 확인할 정수 $K(1 \leq K \leq 10\text{^}50)$가 주어진다.

출력

N개의 줄에 걸쳐 한 줄씩 K가 홀수라면 'odd', 짝수라면 'even'을 출력한다.

예제 입력

```
2
228
123
```

예제 출력

```
even
odd
```

연습문제 02.

문제

영어 대소문자와 공백으로 이루어진 문자열에는 몇 개의 단어가 있는지 구하는 프로그램을 작성하시오. 단, 같은 단어가 여러 번 등장하면 등장 횟수만큼 세어야 한다.

입력

첫째 줄에 영어 대소문자와 공백으로 이루어진 문자열이 주어진다.
단어는 공백 한 개로 구분된다.

출력

첫째 줄에 단어의 개수를 출력한다.

예제 입력

```
Python is known for its simple syntax
```

예제 출력

```
7
```

연습문제 03.

문제

총 N개의 문자열로 이루어진 집합 S가 있다.

입력으로 주어지는 M개의 문자열 중, 집합 S에 포함된 것이 총 몇 개인지 구하는 프로그램을 작성하시오.

입력

첫째 줄에 문자열의 개수 N과 M이 주어진다.

다음 N개의 줄에는 집합 S에 포함된 문자열들이 주어진다.

다음 M개의 줄에는 검사해야 하는 문자열들이 주어진다.

입력으로 주어지는 문자열은 알파벳 소문자로만 이루어져 있으며, 집합 S에 같은 문자열이 여러 번 주어지는 경우는 없다.

출력

첫째 줄에 M개의 문자열 중 총 몇 개가 집합 S에 포함됐는지 출력한다.

예제 입력

```
5 11
baekjoononlinejudge
startlink
codeplus
sundaycoding
codingsh
baekjoon
codeplus
codeminus
startlink
starlink
sundaycoding
codingsh
codinghs
 sondaycoding
startrink
icerink
```

예제 출력

```
4
```

연습문제 04.

문제

괄호 문자열은 두 개의 괄호 기호인 '('와 ')'만으로 구성되어 있는 문자열이다. 그 중 괄호의 모양이 바르게 구성된 문자열을 올바른 괄호 문자열 VPS라고 부른다. 입력으로 주어진 괄호 문자열이 VPS인지 아닌지를 판단하는 프로그램을 작성하시오.

입력

첫째 줄에 입력 데이터의 수를 나타내는 정수 T가 주어진다.

각 테스트 데이터의 첫째 줄에는 괄호 문자열이 한 줄에 주어진다.

하나의 괄호 문자열의 길이는 2 이상 50 이하이다.

출력

만약 VPS가 맞으면 "YES", 아니면 "NO"를 한 줄에 하나씩 차례대로 출력한다.

예제 입력

```
5
((((())))()
((()))((())))
(((()(()))(((())))()
()()()()(()()())()
((()(((())(
```

예제 출력

```
NO
YES
NO
YES
NO
```

연습문제 05.

문제

팰린드롬이란 앞으로 읽을 때와 거꾸로 읽을 때 같은 단어를 말한다.

알파벳 소문자로만 이루어진 단어가 주어졌을 때, 이 단어가 팰린드롬인지 확인하는 프로그램을 작성하시오.

입력

첫째 줄에 단어가 주어진다. 단어는 알파벳 소문자로만 이루어져 있다.

출력

첫째 줄에 팰린드롬이면 “YES“, 아니면 “NO”를 출력한다.

예제 입력	예제 출력
eye	YES
apples	NO

연습문제 06.

문제

땅 위에 있는 달팽이는 높이가 V미터인 나무 막대를 올라간다.

낮 동안에는 A미터를 올라가지만, 밤에 잠을 자는 동안에는 B미터 미끄러진다. 또, 정상에 올라간 뒤에는 미끄러지지 않는다.

달팽이가 나무 막대를 모두 올라가기 위해 며칠이 걸리는지 구하시오.

입력

첫째 줄에 세 정수 A, B, V가 공백으로 구분되어 주어진다.

출력

첫째 줄에 달팽이가 나무 막대를 모두 올라가는데 며칠이 걸리는지 출력한다.

예제 입력

```
2 1 5
```

예제 출력

```
4
```

연습문제 07.

문제

정수를 저장하는 큐를 구현한 다음, 입력으로 주어지는 명령을 처리하는 프로그램을 작성하시오.

- push X: 정수 X를 큐에 넣는 연산이다.
- pop: 큐에서 가장 앞에 있는 정수를 빼고, 그 수를 출력한다. 만약 큐에 들어있는 정수가 없는 경우에는 -1을 출력한다.
- size: 큐에 들어있는 정수의 개수를 출력한다.
- empty: 큐가 비어있으면 1, 아니면 0을 출력한다.
- front: 큐의 가장 앞에 있는 정수를 출력한다. 만약 큐에 들어있는 정수가 없는 경우에는 -1을 출력한다.
- back: 큐의 가장 뒤에 있는 정수를 출력한다. 만약 큐에 들어있는 정수가 없는 경우에는 -1을 출력한다.

입력

첫째 줄에 주어지는 명령의 수 N이 주어진다.

둘째 줄부터 N개의 줄에는 명령이 하나씩 주어진다.

문제에 나와있지 않은 명령이 주어지는 경우는 없다.

출력

출력해야하는 명령이 주어질 때마다, 한 줄에 하나씩 출력한다.

연습문제 07.

예제 입력

```
15
push 1
push 2
front
back
size
empty
pop
pop
pop
size
empty
pop
push 3
empty
front
```

예제 출력

```
1
2
2
0
1
2
-1
0
1
-1
0
3
```

연습문제 08.

문제

프린터 기기는 인쇄 명령을 받은 순서대로 명령을 수행한다. 여러 개의 문서가 쌓인다면 Queue 자료구조에 쌓여 FIFO에 따라 인쇄가 된다. 하지만 길동이는 새로운 프린터기 소프트웨어를 개발하였는데, 이 프린터기는 아래의 조건에 따라 인쇄를 하게 된다.

1. 현재 Queue의 가장 앞에 있는 문서의 '중요도'를 확인하다.
2. 나머지 문서들 중 현재 문서보다 중요도가 높은 문서가 하나라도 있다면, 이 문서를 인쇄하지 않고 Queue의 가장 뒤에 재배치 한다. 그렇지 않다면 바로 인쇄를 한다.

예를 들어, Queue에 4개의 문서(A B C D)가 있고, 중요도가 2 1 4 3 라면 C를 인쇄하고, 다음으로 D를 인쇄하고 A, B를 인쇄하게 된다.

현재 Queue에 있는 문서의 수와 중요도가 주어졌을 때, 어떤 한 문서가 몇 번째로 인쇄되는지 알아내는 프로그램을 작성하시오.

입력

첫 줄에 테스트케이스의 수가 주어진다.

각 테스트케이스는 두 줄로 이루어져 있다.

테스트케이스의 첫 번째 줄에는 문서의 개수 N과, 몇 번째로 인쇄되었는지 궁금한 문서가 현재 Queue에서 몇 번째에 놓여 있는지를 나타내는 정수 M이 주어진다. 이때 맨 왼쪽은 0번째라고 하자. 두 번째 줄에는 N개 문서의 중요도가 차례대로 주어진다. 중요도는 1 이상 9 이하의 정수이고, 중요도가 같은 문서가 여러 개 있을 수도 있다.

연습문제 08.

출력

각 테스트 케이스에 대해 문서가 몇 번째로 인쇄되는지 출력한다.

예제 입력

```
3
1 0
5
4 2
1 2 3 4
6 0
1 1 9 1 1 1
```

예제 출력

```
1
2
5
```

연습문제 09.

문제

정수를 저장하는 스택을 구현한 다음, 입력으로 주어지는 명령을 처리하는 프로그램을 작성하시오.

- push X: 정수 X를 스택에 넣는 연산이다.
- pop: 스택에서 가장 위에 있는 정수를 빼고, 그 수를 출력한다. 만약 스택에 들어있는 정수가 없는 경우에는 -1을 출력한다.
- size: 스택에 들어있는 정수의 개수를 출력한다.
- empty: 스택이 비어있으면 1, 아니면 0을 출력한다.
- top: 스택의 가장 위에 있는 정수를 출력한다. 만약 스택에 들어있는 정수가 없는 경우에는 -1을 출력한다.

입력

첫째 줄에 주어지는 명령의 수 N이 주어진다.

둘째 줄부터 N개의 줄에는 명령이 하나씩 주어진다.

문제에 나와있지 않은 명령이 주어지는 경우는 없다.

출력

출력해야하는 명령이 주어질 때마다, 한 줄에 하나씩 출력한다.

연습문제 09.

예제 입력

```
7
pop
top
push 123
top
pop
top
pop
```

예제 출력

```
-1
-1
123
123
-1
-1
```

연습문제 10.

문제

후위 표기법(Reverse Polish Notation, RPN)으로 표현된 수식이 주어진다. 후위 표기법은 연산자가 피연산자 뒤에 위치하는 표기법이다. 주어진 후위 표기법을 계산하여 결과를 출력하시오.

후위 표기법 규칙

1. 피연산자는 순서대로 스택에 쌓인다.
2. 연산자가 나오면 스택의 상위 두 개의 피연산자를 꺼내 연산을 수행한 후, 그 결과를 다시 스택에 넣는다.
3. 모든 연산이 끝난 후, 스택에 남은 값이 결과이다.

연산자는 +, -, *, /만 사용할 수 있으며, 입력된 값은 정수이다.

나눗셈의 경우, 항상 정수 나눗셈으로 처리되며 결과는 정수로 반환된다.

입력

첫째 줄에 후위 표기법으로 이루어진 수식이 공백으로 구분되어 주어진다.

수식은 최대 100개의 피연산자 및 연산자로 구성된다.

피연산자는 1 이상 100 이하의 정수이다.

출력

주어진 후위 표기법 수식을 계산한 결과를 출력한다.

연습문제 10.

예제 입력 1	5 1 2 + 4 * + 3 -	예제 출력 1	14
예제 입력 2	2 3 * 5 4 * + 9 -	예제 출력 2	17

연습문제 11.

문제

2차원 평면 위의 N개의 점의 좌표를 y좌표가 증가하는 순으로 y좌표가 같으면 x좌표가 증가하는 순서로 정렬한 다음 출력하시오,

입력

첫째 줄에 점의 개수 $N(1 \le N \le 100{,}000)$이 주어진다.

둘째 줄부터 N개의 줄에는 i번 점의 위치 X_i와 Y_i가 주어진다.

좌표는 항상 정수이고, 위치가 같은 두 점은 없다.

$(-100{,}000 \le X_i, Y_i \le 100{,}000)$

출력

첫째 줄부터 N개의 줄에 점을 정렬한 결과를 출력한다.

예제 입력

```
5
0 4
1 2
1 -1
2 2
3 3
```

예제 출력

```
1 -1
1 2
2 2
3 3
0 4
```

연습문제 12.

문제

숫자카드 N개와 정수 M개가 주어졌을 때 이 수가 적혀 있는 숫자 카드를 몇 개 가지고 있는지 구하는 프로그램을 작성하시오.

입력

첫째 줄에 숫자 카드의 개수 $N(1 \leq N \leq 500{,}000)$이 주어진다.

둘째 줄에는 숫자 카드에 적혀있는 정수가 주어진다.

셋째 줄에는 $M(1 \leq M \leq 500{,}000)$이 주어진다.

넷째 줄에는 몇 개 가지고 있는 숫자 카드 인지 구해야 할 M개의 정수가 주어지며, 이 수는 공백으로 구분되어져 있다.

출력

첫째 중에 입력으로 주어진 M개의 수에 대해서, 각 수가 적힌 숫자 카드가 존재하면 1을, 아니면 0을 공백으로 구분해 출력한다.

예제 입력

```
10
6 3 2 10 10 10 -10 -10 7 3
8
10 9 -5 2 3 4 5 -10
```

예제 출력

```
3 0 0 1 2 0 0 2
```

연습문제 13.

문제

수직선 위에 $N(1 \le N \le 1{,}000{,}000)$개의 좌표 $X_1, X_2, \dots, X_N$이 있다. 이 좌표에 좌표 압출을 적용하려고 한다. $X_i(-109 \le X_{\mathrm{i}} \le 109)$를 좌표 압축한 결과 X_i'의 값은 $X_i > X_j$를 만족하는 서로 다른 좌표의 개수와 같아야 한다.

$X_1, X_2, \dots, X_N$에 좌표 압축을 적용한 결과 $X_1', X_2', \dots, X_N'$를 출력해보자.

입력

첫째 줄에 N이 주어진다.

둘째 줄에는 공백 한칸으로 구분된 $X_1, X_2, \dots, X_N$이 주어진다.

출력

첫째 줄에 $X_1', X_2', \dots, X_N'$을 공백 한 칸으로 구분해서 출력한다.

예제 입력 1

```
5
2 4 -10 4 -9
```

예제 출력 1

```
2 3 0 3 1
```

예제 입력 2

```
6
1000 999 1000 999 1000 999
```

예제 출력 2

```
1 0 1 0 1 0
```

연습문제 14.

문제

알파벳 소문자로 이루어진 $N(1 \leq N \leq 20{,}000)$개의 단어가 들어오면 다음과 같은 조건에 따라 정렬하시오. 길이가 짧은 것부터, 길이가 같으면 사전 순으로 정렬하시오. 단, 중복된 단어는 하나만 남기고 제거해야 한다.

입력

첫째 줄에 N이 주어진다.

둘째 줄부터 N개의 줄에 걸쳐 알파벳 소문자로 이루어진 단어가 한 줄에 하나씩 주어진다. 주어지는 문자열의 길이는 50을 넘지 않는다.

출력

조건에 따라 정렬하여 단어들을 출력한다. 단, 같은 단어가 여러 번 입력된 경우에는 한번식만 출력한다.

예제 입력

```
13
but
i
wont
hesitate
no
more
no
more
it
cannot
wait
im
yours
```

예제 출력

```
i
im
it
no
but
more
wait
wont
yours
cannot
hesitate
```

연습문제 15.

문제

집합의 어떠한 단어가 다른 단어의 접두어가 되지 않는 집합의 최대 크기를 구하시오.

{hello}, {hello, goodbye, giant, hi} : O

{hello, hell}, {giant, gig, g} : X

입력

첫째 줄에 $N(1 \leq N \leq 50)$이 주어진다.

둘째 줄부터 N 개의 줄에는 단어가 주어진다. 단어는 알파벳 소문자로만 이루어져 있고, 길이는 최대 50이다. 집합에는 같은 단어가 두 번 이상 있을 수 있다.

출력

첫째 줄에 문제의 정답을 출력한다.

예제 입력

```
6
hello
hi
h
run
rerun
running
```

예제 출력

```
4
```

연습문제 16.

문제

원소의 개수가 N개이고 값이 서로 다른 정수 배열 A를 오름차순으로 만들고 싶다. 배열의 i번째 원소와 i+1번째 원소끼리 서로 위치를 바꿀 수 있고, 정렬 과정 중 언제든지 최대 딱 한 번 배열 전체의 순서를 뒤집을 수 있다. 원소를 교환하는 것, 배열 전체를 뒤집는 것 모두 1번의 횟수로 계산한다. 주어진 배열 A를 오름차순으로 만드는데 필요한 최소한의 횟수를 구하여라.

입력

첫 번째 줄에 배열 A의 원소 개수 $N(1 \leq N \leq 50)$이 주어진다.

두 번째 줄에 A의 원소 정수 A_i가 공백을 사이에 두고 순서대로 주어진다.

출력

첫 번째 줄에 주어진 배열 A를 오름차순으로 만드는데 필요한 최소한의 횟수를 출력한다.

예제 입력

```
5
2 5 4 1 3
```

예제 출력

```
6 10 2
```

연습문제 17.

문제

배열의 array의 i번째 숫자부터 j번째 숫자까지 자르고 정렬했을 때, k번째에 있는 수를 구하려 한다.

예를 들어 array가 [1, 5, 2, 6, 3, 7, 4], i = 2, j = 5, k = 3이라면

1. array의 2번째부터 5번째까지 자르면 [5, 2, 6, 3]이다.
2. 1에서 나온 배열을 정렬하면 [2, 3, 5, 6]이다.
3. 2에서 나온 배열의 3번째 숫자는 5이다.

입력

배열 array, [i, j, k]를 원소로 가진 2차원 배열 commands가 매개변수로 주어진다.

출력

commands의 모든 원소에 대해 앞서 설명한 연산을 적용했을 때 나온 결과를 출력한다.

예제 입력

```
array = [1, 5, 2, 6, 3, 7, 4]
Commands = [[2, 5, 3], [4, 4, 1], [1, 7, 3]]
```

예제 출력

```
[5, 6, 3]
```

연습문제 18.

문제

0 또는 양의 정수가 주어졌을 때, 정수를 이어 붙여 만들 수 있는 가장 큰 수를 출력하시오.

예를 들어, 주어진 정수가 [6, 10, 20]이라면 [6102, 6210, 1062, 1026, 2610, 2106]를 만들 수 있고, 이 중 가장 큰 수는 6210이다.

입력

0 또는 양의 정수가 담긴 배열 numbers가 매개변수로 주어진다.

출력

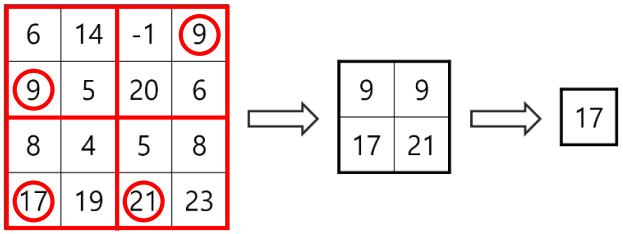
순서를 재배치하여 만들 수 있는 가장 큰 수를 문자열로 바꾸어 출력한다.

예제 입력 1	6 10 2	예제 출력 1	“6210”
예제 입력 2	3 30 34 5 9	예제 출력 2	“9534330”

연습문제 19.

문제

행렬을 2x2 정사각형으로 나눈 후, 각 정사각형에서 2번째로 큰 수만 남긴다. 여기서 2번째로 큰 수란, 정사각형의 네 원소를 크기순으로 $a_4 \leq a_3 \leq a_2 \leq a_1$이라 했을 때, 원소 a_2를 뜻한다. 마지막에 남은 수를 출력한다.



입력

첫째 줄에 $N(2 \leq N \leq 1024)$이 주어진다. N은 항상 2의 거듭제곱 꼴이다. $(N = 2K,\ 1 \leq K \leq 10)$ 다음 N개의 줄마다 각 행의 원소 N개가 차례대로 주어진다. 행렬의 모든 성분은 -10,000 이상 10,000이하의 정수이다.

출력

마지막에 남은 수를 출력한다.

예제 입력

```
4
-6 -8 7 -4
-5 -5 14 11
11 11 -1 -1
4 9 -2 -4
```

예제 출력

```
11
```

연습문제 20.

문제

세개의 장대가 있고 첫 번째 장대에는 반경이 서로 다른 n개의 원판이 쌓여 있다. 각 원판은 반경이 큰 순서대로 쌓여있다. 다음 규칙에 따라 첫 번째 장대에서 세 번째 장대로 옮기려 한다.

1. 한 번에 한 개의 원판만을 다른 탑으로 옮길 수 있다.

2. 쌓아 놓은 원판은 항상 위의 것이 아래의 것보다 작아야 한다.

이 작업을 수행하는데 필요한 이동 순서를 출력하는 프로그램을 작성하라. 단, 이동 횟수는 최소가 되어야 한다.

입력

첫째 줄에 첫 번째 장대에 쌓인 원판의 개수 $N(1 \le N \le 20)$이 주어진다.

출력

첫째 줄에 옮긴 횟수 K를 출력한다. 두 번째 줄부터 수행 과정을 출력한다. 두 번째 줄부터 K개의 걸쳐 두 정수 A B를 빈칸을 사이에 두고 출력하는데 이는 A번째 탑의 가장 위에 있는 원판을 B번째 탑의 가장 위로 옮긴다는 뜻이다.

예제 입력

```
3
```

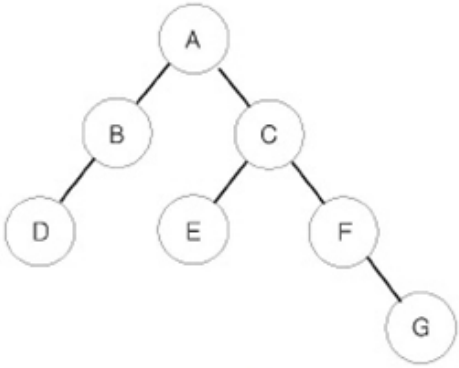
예제 출력

```
7
1 3
1 2
3 2
1 3
2 1
2 3
1 3
```

연습문제 21.

문제

이진트리를 입력받아 전위 순회(preorder traversal), 중위 순회(inorder traversal), 후위 순회(postorder traversal)한 결과를 출력하는 프로그램을 작성하시오. 예를 들어 이와 같은 트리가 입력 되면,



전위 순회한 결과 : ABDCEFG //(루트) (왼쪽 자식) (오른쪽 자식)

중위 순회한 결과 : DBAECFG // (왼쪽 자식) (루트) (오른쪽 자식)

후위 순회한 결과 : DBEGFCA // (왼쪽 자식) (오른쪽 자식) (루트)

가 된다.

연습문제 21.

입력

첫째 줄에는 이진 트리의 노드의 개수 N(1 N 26)이 주어진다. 둘째 줄부터 N개의 줄에 걸쳐 각 노드와 그의 왼쪽 자식 노드, 오른쪽 자식 노드가 주어진다. 노드의 이름은 A부터 차례대로 알파벳 대문자로 매겨지며, 항상 A가 루트 노드가 된다. 자식 노드가 없는 경우에는 .으로 표현한다.

출력

첫째 줄에 전위 순회, 둘째 줄에 중위 순회, 셋째 줄에 후위 순회한 결과를 출력한다. 각 줄에 N개의 알파벳을 공백 없이 출력하면 된다.

예제 입력1

```
7
A B C
B D .
C E F
E . .
F . G
D . .
G . .
```

예제 출력1

```
ABDCEFG
DBAECFG
DBEGFCA
```

연습문제 22.

문제

루트 없는 트리가 주어진다. 이때, 트리의 루트를 1이라고 정했을 때, 각 노드의 부모를 구하는 프로그램을 작성하시오.

입력

첫째 줄에 노드의 개수 N (2 N 100,000)이 주어진다. 둘째 줄부터 N-1개의 줄에 트리 상에서 연결된 두 정점이 주어진다.

출력

첫째 줄부터 N-1개의 줄에 각 노드의 부모 노드 번호를 2번 노드부터 순서대로 출력한다.

예제 입력1

```
7
1 6
6 3
3 5
4 1
2 4
4 7
```

예제 출력1

```
4
6
1
3
1
4
```

예제 입력2

```
12
1 2
1 3
2 4
3 5
3 6
4 7
4 8
5 9
5 10
6 11
6 12
```

예제 출력2

```
1
1
2
3
3
4
4
5
5
6
6
```

연습문제 23.

문제

그래프를 DFS로 탐색한 결과와 BFS로 탐색한 결과를 출력하는 프로그램을 작성하시오. 단, 방문할 수 있는 정점이 여러 개인 경우에는 정점 번호가 작은 것을 먼저 방문하고, 더 이상 방문할 수 있는 점이 없는 경우 종료한다. 정점 번호는 1번부터 N번까지이다.

입력

첫째 줄에 정점의 개수 N(1 N 1,000), 간선의 개수 M(1 M 10,000), 탐색을 시작할 정점의 번호 V가 주어진다. 다음 M개의 줄에는 간선이 연결하는 두 정점의 번호가 주어진다. 어떤 두 정점 사이에 여러 개의 간선이 있을 수 있다. 입력으로 주어지는 간선은 양방향이다.

출력

첫째 줄에 DFS를 수행한 결과를, 그 다음 줄에는 BFS를 수행한 결과를 출력한다. V부터 방문된 점을 순서대로 출력하면 된다.

예제 입력1

```
4 5 1
1 2
1 3
1 4
2 4
3 4
```

예제 출력1

```
1 2 4 3
1 2 3 4
```

예제 입력2

```
5 5 3
5 4
5 2
1 2
3 4
3 1
```

예제 출력2

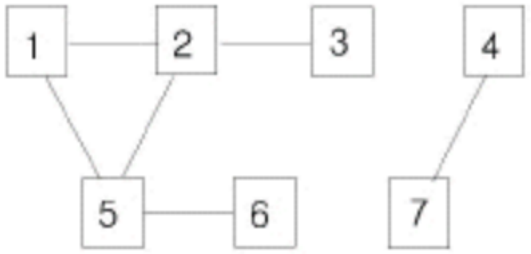
```
3 1 2 5 4
3 1 4 2 5
```

연습문제 24.

문제

신종 바이러스인 웜 바이러스는 네트워크를 통해 전파된다. 한 컴퓨터가 웜 바이러스에 걸리면 그 컴퓨터와 네트워크 상에서 연결되어 있는 모든 컴퓨터는 웜 바이러스에 걸리게 된다.

예를 들어 7대의 컴퓨터가 <그림 1>과 같이 네트워크 상에서 연결되어 있다고 하자. 1번 컴퓨터가 웜 바이러스에 걸리면 웜 바이러스는 2번과 5번 컴퓨터를 거쳐 3번과 6번 컴퓨터까지 전파되어 2, 3, 5, 6 네 대의 컴퓨터는 웜 바이러스에 걸리게 된다. 하지만 4번과 7번 컴퓨터는 1번 컴퓨터와 네트워크상에서 연결되어 있지 않기 때문에 영향을 받지 않는다.



< 그림 1 >

어느 날 1번 컴퓨터가 웜 바이러스에 걸렸다. 컴퓨터의 수와 네트워크 상에서 서로 연결되어 있는 정보가 주어질 때, 1번 컴퓨터를 통해 웜 바이러스에 걸리게 되는 컴퓨터의 수를 출력하는 프로그램을 작성하시오.

연습문제 24.

입력

첫째 줄에는 컴퓨터의 수가 주어진다. 컴퓨터의 수는 100 이하인 양의 정수이고 각 컴퓨터에는 1번 부터 차례대로 번호가 매겨진다. 둘째 줄에는 네트워크 상에서 직접 연결되어 있는 컴퓨터 쌍의 수가 주어진다. 이어서 그 수만큼 한 줄에 한 쌍씩 네트워크 상에서 직접 연결되어 있는 컴퓨터의 번호 쌍이 주어진다.

출력

1번 컴퓨터가 웜 바이러스에 걸렸을 때, 1번 컴퓨터를 통해 웜 바이러스에 걸리게 되는 컴퓨터의 수를 첫째 줄에 출력한다.

예제 입력1

```
7
6
1 2
2 3
1 5
5 2
5 6
4 7
```

예제 출력1

```
4
```

연습문제 25.

문제

방향 없는 그래프가 주어졌을 때, 연결 요소 (Connected Component)의 개수를 구하는 프로그램을 작성하시오.

입력

첫째 줄에 정점의 개수 N과 간선의 개수 M이 주어진다. (1 N 1,000, 0 M N (N-1)/2) 둘째 줄부터 M개의 줄에 간선의 양 끝점 u와 v가 주어진다. (1 u, v N, u v) 같은 간선은 한 번만 주어진다.

출력

첫째 줄에 연결 요소의 개수를 출력한다.

예제 입력1

```
6 5
1 2
2 5
5 1
3 4
4 6
```

예제 출력1

```
2
```

예제 입력2

```
6 8
1 2
2 5
5 1
3 4
4 6
5 4
2 4
2 3
```

예제 출력2

```
1
```

연습문제 26.

문제

방향그래프가 주어지면 주어진 시작점에서 다른 모든 정점으로의 최단 경로를 구하는 프로그램을 작성하시오. 단, 모든 간선의 가중치는 10 이하의 자연수이다.

입력

첫째 줄에 정점의 개수 V와 간선의 개수 E가 주어진다. (1 V 20,000, 1 E 300,000) 모든 정점에는 1부터 V까지 번호가 매겨져 있다고 가정한다. 둘째 줄에는 시작 정점의 번호 K(1 K V)가 주어진다. 셋째 줄부터 E개의 줄에 걸쳐 각 간선을 나타내는 세 개의 정수 (u, v, w)가 순서대로 주어진다. 이는 u에서 v로 가는 가중치 w인 간선이 존재한다는 뜻이다. u와 v는 서로 다르며 w는 10 이하의 자연수이다. 서로 다른 두 정점 사이에 여러 개의 간선이 존재할 수도 있음에 유의한다.

출력

첫째 줄부터 V개의 줄에 걸쳐, i번째 줄에 i번 정점으로의 최단 경로의 경로값을 출력한다. 시작점 자신은 0으로 출력하고, 경로가 존재하지 않는 경우에는 INF를 출력하면 된다.

연습문제 26.

예제 입력1

```
5 6
1
5 1 1
1 2 2
1 3 3
2 3 4
2 4 5
3 4 6
```

예제 출력1

```
0
2
3
7
INF
```

연습문제 27.

문제

월드 나라는 모든 도로가 일방통행인 도로이고, 싸이클이 없다. 그런데 어떤 무수히 많은 사람들이 월드 나라의 지도를 그리기 위해서, 어떤 시작 도시로부터 도착 도시까지 출발을 하여 가능한 모든 경로를 탐색한다고 한다.

이 지도를 그리는 사람들은 사이가 너무 좋아서 지도를 그리는 일을 다 마치고 도착 도시에서 모두 다 만나기로 하였다. 그렇다고 하였을 때 이들이 만나는 시간은 출발 도시로부터 출발한 후 최소 몇 시간 후에 만날 수 있는가? 즉, 마지막에 도착하는 사람까지 도착을 하는 시간을 의미한다. 어떤 사람은 이 시간에 만나기 위하여 1분도 쉬지 않고 달려야 한다. 이런 사람들이 지나는 도로의 수를 카운트 하여라.

출발 도시는 들어오는 도로가 0개이고, 도착 도시는 나가는 도로가 0개이다.

입력

첫째 줄에 도시의 개수 n(1 n 10,000)이 주어지고 둘째 줄에는 도로의 개수 m(1 m 100,000)이 주어진다. 그리고 셋째 줄부터 m+2줄까지 다음과 같은 도로의 정보가 주어진다. 처음에는 도로의 출발 도시의 번호가 주어지고 그 다음에는 도착 도시의 번호, 그리고 마지막에는 이 도로를 지나는데 걸리는 시간이 주어진다. 도로를 지나가는 시간은 10,000보다 작거나 같은 자연수이다.

그리고 m+3째 줄에는 지도를 그리는 사람들이 출발하는 출발 도시와 도착 도시가 주어진다.

연습문제 27.

모든 도시는 출발 도시로부터 도달이 가능하고, 모든 도시로부터 도착 도시에 도달이 가능하다.

출력

첫째 줄에는 이들이 만나는 시간을, 둘째 줄에는 1분도 쉬지 않고 달려야 하는 도로의 수가 몇 개인지 출력하여라.

예제 입력1

```
7
9
1 2 4
1 3 2
1 4 3
2 6 3
2 7 5
3 5 1
4 6 4
5 6 2
6 7 5
1 7
```

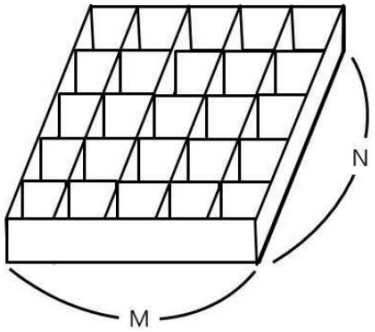
예제 출력1

```
12
5
```

연습문제 28.

문제

철수의 토마토 농장에서는 토마토를 보관하는 큰 창고를 가지고 있다. 토마토는 아래의 그림과 같이 격자 모양 상자의 칸에 하나씩 넣어서 창고에 보관한다.



창고에 보관되는 토마토들 중에는 잘 익은 것도 있지만, 아직 익지 않은 토마토들도 있을 수 있다. 보관 후 하루가 지나면, 익은 토마토들의 인접한 곳에 있는 익지 않은 토마토들은 익은 토마토의 영향을 받아 익게 된다. 하나의 토마토의 인접한 곳은 왼쪽, 오른쪽, 앞, 뒤 네 방향에 있는 토마토를 의미한다. 대각선 방향에 있는 토마토들에게는 영향을 주지 못하며, 토마토가 혼자 저절로 익는 경우는 없다고 가정한다. 철수는 창고에 보관된 토마토들이 며칠이 지나면 다 익게 되는지, 그 최소 일수를 알고 싶어 한다. 토마토를 창고에 보관하는 격자모양의 상자들의 크기와 익은 토마토들과 익지 않은 토마토들의 정보가 주어졌을 때, 며칠이 지나면 토마토들이 모두 익는지, 그 최소 일수를 구하는 프로그램을 작성하라. 단, 상자의 일부 칸에는 토마토가 들어있지 않을 수도 있다.

연습문제 28.

입력

첫 줄에는 상자의 크기를 나타내는 두 정수 M,N이 주어진다. M은 상자의 가로 칸의 수, N은 상자의 세로 칸의 수를 나타낸다. 단, 2 M,N 1,000 이다. 둘째 줄부터는 하나의 상자에 저장된 토마토들의 정보가 주어진다. 즉, 둘째 줄부터 N개의 줄에는 상자에 담긴 토마토의 정보가 주어진다. 하나의 줄에는 상자 가로줄에 들어있는 토마토의 상태가 M개의 정수로 주어진다. 정수 1은 익은 토마토, 정수 0은 익지 않은 토마토, 정수 -1은 토마토가 들어있지 않은 칸을 나타낸다.

토마토가 하나 이상 있는 경우만 입력으로 주어진다.

출력

여러분은 토마토가 모두 익을 때까지의 최소 날짜를 출력해야 한다. 만약, 저장될 때부터 모든 토마토가 익어있는 상태이면 0을 출력해야 하고, 토마토가 모두 익지는 못하는 상황이면 -1을 출력해야 한다.

예제 입력1

```
6 4
0 0 0 0 0 0
0 0 0 0 0 0
0 0 0 0 0 0
0 0 0 0 0 1
```

예제 출력1

```
8
```

연습문제 28.

예제 입력2

```
6 4
0 -1 0 0 0 0
-1 0 0 0 0 0
0 0 0 0 0 0
0 0 0 0 0 1
```

예제 출력2

```
-1
```

예제 입력3

```
6 4
1 -1 0 0 0 0
0 -1 0 0 0 0
0 0 0 0 -1 0
0 0 0 0 -1 1
```

예제 출력3

```
6
```

예제 입력4

```
5 5
-1 1 0 0 0
0 -1 -1 -1 0
0 -1 -1 -1 0
0 -1 -1 -1 0
0 0 0 0 0
```

예제 출력4

```
14
```

연습문제 29.

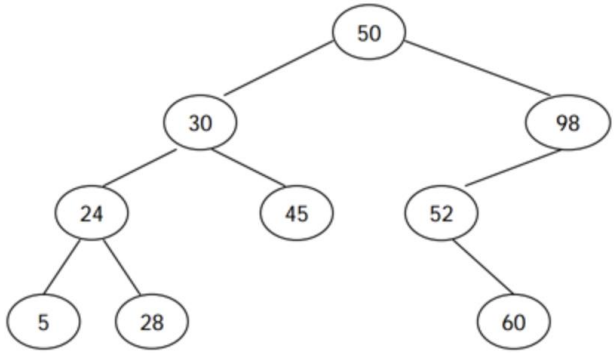
문제

이진 검색 트리는 다음과 같은 세 가지 조건을 만족하는 이진 트리이다.

•노드의 왼쪽 서브트리에 있는 모든 노드의 키는 노드의 키보다 작다.

•노드의 오른쪽 서브트리에 있는 모든 노드의 키는 노드의 키보다 크다.

•왼쪽, 오른쪽 서브트리도 이진 검색 트리이다.



전위 순회 (루트-왼쪽-오른쪽)은 루트를 방문하고, 왼쪽 서브트리, 오른쪽 서브 트리를 순서대로 방문하면서 노드의 키를 출력한다. 후위 순회 (왼쪽-오른쪽-루트)는 왼쪽 서브트리, 오른쪽 서브트리, 루트 노드 순서대로 키를 출력한다. 예를 들어, 위의 이진 검색 트리의 전위 순회 결과는 50 30 24 5 28 45 98 52 60 이고, 후위 순회 결과는 5 28 24 45 30 60 52 98 50 이다.

이진 검색 트리를 전위 순회한 결과가 주어졌을 때, 이 트리를 후위 순회한 결과를 구하는 프로그램을 작성하시오.

연습문제 29.

입력

트리를 전위 순회한 결과가 주어진다. 노드에 들어있는 키의 값은 10^6보다 작은 양의 정수이다. 모든 값은 한 줄에 하나씩 주어지며, 노드의 수는 10,000개 이하이다. 같은 키를 가지는 노드는 없다.

출력

입력으로 주어진 이진 검색 트리를 후위 순회한 결과를 한 줄에 하나씩 출력한다.

예제 입력1

```
50
30
24
5
28
45
98
52
60
```

예제 출력1

```
5
28
24
45
30
60
52
98
50
```

연습문제 30.

문제

N개의 도시가 있다. 그리고 한 도시에서 출발하여 다른 도시에 도착하는 M개의 버스가 있다. 우리는 A번째 도시에서 B번째 도시까지 가는데 드는 버스 비용을 최소화 시키려고 한다. A번째 도시에서 B번째 도시까지 가는데 드는 최소비용을 출력하여라. 도시의 번호는 1부터 N까지이다.

입력

첫째 줄에 도시의 개수 N(1 N 1,000)이 주어지고 둘째 줄에는 버스의 개수 M(1 M 100,000)이 주어진다. 그리고 셋째 줄부터 M+2줄까지 다음과 같은 버스의 정보가 주어진다. 먼저 처음에는 그 버스의 출발 도시의 번호가 주어진다. 그리고 그 다음에는 도착지의 도시 번호가 주어지고 또 그 버스 비용이 주어진다. 버스 비용은 0보다 크거나 같고, 100,000보다 작은 정수이다.

그리고 M+3째 줄에는 우리가 구하고자 하는 구간 출발점의 도시번호와 도착점의 도시번호가 주어진다. 출발점에서 도착점을 갈 수 있는 경우만 입력으로 주어진다.

출력

첫째 줄에 출발 도시에서 도착 도시까지 가는데 드는 최소 비용을 출력한다.

연습문제 30.

예제 입력1

```
5
8
1 2 2
1 3 3
1 4 1
1 5 10
2 4 2
3 4 1
3 5 1
4 5 3
1 5
```

예제 출력1

```
4
```

연습문제 01.

```python
n = int(input())

for _ in range(n):
    number = input().strip()

    if int(number[-1]) % 2 == 0:
        print("even")
    else:
        print("odd")
```

연습문제 02.

```python
sentence = input().strip()

words = sentence.split()

print(len(words))
```

연습문제 03.

```python
n, m = map(int, input().split())

s_set = set()

for _ in range(n):
    s_set.add(input().strip())

count = 0
for _ in range(m):
    check_string = input().strip()
    if check_string in s_set:
        count += 1

print(count)
```

연습문제 04.

```
t = int(input())

for _ in range(t):
    ps = input().strip()
    stack = []
    is_vps = True

    for char in ps:
        if char == '(':
            stack.append(char)
        elif char == ')':
            if stack:
                stack.pop()
            else:
                is_vps = False
                break

        if is_vps and not stack:
        print("YES")
    else:
        print("NO")
```

연습문제 05.

```
word = input().strip()

if word == word[::-1]:
    print("YES")
else:
    print("NO")
```

연습문제 06.

```python
A, B, V = map(int, input().split())

days = (V - B) // (A - B)
if (V - B) % (A - B) != 0:
    days += 1

print(days)
```

연습문제 07.

```python
from collections import deque
import sys
input = sys.stdin.readline

queue = deque()

n = int(input())

for _ in range(n):
    command = input().strip()

    if command.startswith("push"):
        _, num = command.split()
        queue.append(num)

    elif command == "pop":
        if queue:
            print(queue.popleft())
        else:
            print(-1)

    elif command == "size":
        print(len(queue))

    elif command == "empty":
        print(1 if not queue else 0)

    elif command == "front":
        if queue:
            print(queue[0])
        else:
            print(-1)

    elif command == "back":
        if queue:
            print(queue[-1])
        else:
            print(-1)
```

연습문제 08.

```python
from collections import deque

t = int(input())

for _ in range(t):
    n, m = map(int, input().split())

    priorities = list(map(int, input().split()))
    queue = deque((priority, idx) for idx, priority in enumerate(priorities))

    count = 0

    while queue:
        current = queue.popleft()

        if any(current[0] < q[0] for q in queue):
            queue.append(current)
        else:
            count += 1
            if current[1] == m:
                print(count)
                break

```

연습문제 09.

```python
import sys
input = sys.stdin.readline

stack = []

n = int(input())

for _ in range(n):
    command = input().strip()

    if command.startswith("push"):
        _, num = command.split()
        stack.append(int(num))

    elif command == "pop":
        if stack:
            print(stack.pop())
        else:
            print(-1)

    elif command == "size":
        print(len(stack))

    elif command == "empty":
        print(1 if not stack else 0)

    elif command == "top":
        if stack:
            print(stack[-1])
        else:
            print(-1)
```

연습문제 10.

```
def postfix(expression):
    stack = []
    for token in expression.split():
        if token.isdigit():
            stack.append(int(token))
        else:
            b = stack.pop()
            a = stack.pop()
            if token == '+':
                stack.append(a + b)
            elif token == '-':
                stack.append(a - b)
            elif token == '*':
                stack.append(a * b)
            elif token == '/':
                stack.append(a // b)

    return stack[0]
expression = input().strip()
result = postfix(expression)
print(result)
```

연습문제 11.

```
n = int(input())

arr = []

for i in range(n):
    arr.append(list(map(int, input().split())))

arr = sorted(arr)

for i in range(n):
    print(arr[i][0], arr[i][1])
```

연습문제 12.

```
_ = int(input())
N = list(map(int, input().split()))
_ = int(input())
M = list(map(int, input().split()))
index, m_dic = 0, {}

sorted_M = sorted(M)
N.sort()

for m in sorted_M:
    cnt = 0
    if m not in m_dic:
        while index < len(N):
            if m == N[index]:
                cnt+=1; index+=1
            elif m > N[index]:
                index += 1
            else: break
        m_dic[m] = cnt

print(' '.join(str(m_dic[m]) for m in M))
```

연습문제 13.

```
n = int(input())
array = list(map(int, input().split()))

array2 = list(set(array))
array2.sort()
dic = {array2[i] : i for i in range(len(array2))}

for i in array:
    print(dic[i], end = ' ')
```

연습문제 14.

```python
N = int(input())

array = [None for i in range(N)]

for i in range(N):
    array[i] = str(input())

array = list(set(array))

array = sorted(array)
array = sorted(array, key = len)

for i in range(len(array)):
    print(array[i])
```

연습문제 15.

```python
N = int(input())
words = [input().rstrip() for _ in range(N)]

words.sort(key = len)
res = 0

for i in range(N):
    flag = False
    for j in range(i+1, N):
        if words[i] == words[j][0:len(words[i])]:
            flag = True
            break

    if not flag:
        res += 1

print(res)
```

연습문제 16.

```
N = int(input())

array = list(map(int,input().split()))
array2 = array.copy()

def ascend(arr):
    swap_cnt = 0
    for i in range(len(arr)-1, 0, -1):
        for j in range(i):
            if arr[j] > arr[j + 1]:
                arr[j], arr[j+1] = arr[j+1], arr[j]
                swap_cnt += 1
    return swap_cnt

def descend(arr):
    swap_cnt = 0
    for i in range(len(arr)-1, 0, -1):
        for j in range(i):
            if arr[j] < arr[j + 1]:
                arr[j], arr[j+1] = arr[j+1], arr[j]
                swap_cnt += 1
    return swap_cnt + 1

print(min(ascend(array), descend(array2)))
```

연습문제 17.

```
array = [1, 5, 2, 6, 3, 7, 4]
commands = [[2, 5, 3], [4, 4, 1], [1, 7, 3]]
answer = []

for i in range(len(commands)):
    array1 = sorted(array[commands[i][0]-1:commands[i][1]])
    answer.append(array1[commands[i][2]-1])

print(answer)
```

연습문제 18.

```
numbers = list(map(str, input().split()))
numbers.sort(key=lambda x : x*3, reverse = True)
print(str(it(''.join(numbers))))
```

연습문제 19.

```
N = int(input())

array = [[] for _ in range(N)]

for i in range(N):
    array[i] = list(map(int, input().split()))

def pooling(arr):
    new_arr = [[] for _ in range(len(arr)//2)]
    for i in range(0, len(arr), 2):
        for j in range(0, len(arr), 2):
            sorted_arr = [arr[i][j], arr[i+1][j], arr[i][j+1], arr[i+1][j+1]]
            sorted_arr.sort()
            new_arr[i//2].append(sorted_arr[-2])
    if len(new_arr) == 1:
        return new_arr[0][0]
    else:
        return pooling(new_arr)

print(pooling(array))
```

연습문제 20.

```
N = int(input())

print(2**N-1)

def hanoi(number_of_disks_to_move, from_, to_, via_):
    if number_of_disks_to_move == 1:
        print(from_, to_)
    else:
        hanoi(number_of_disks_to_move-1, from_, via_, to_)
        print(from_, to_)
        hanoi(number_of_disks_to_move-1, via_, to_, from_)

hanoi(N,1,3,2)
```

연습문제 21.

```
import sys;

def preOrder(node):
        if node == None:
                return;
        print(node, end="");
        preOrder(edgeList[node][0]);
        preOrder(edgeList[node][1]);
def inOrder(node):
        if node == None:
                return;
        inOrder(edgeList[node][0]);
        print(node, end= "");
        inOrder(edgeList[node][1]);
def postOrder(node):
        if node == None:
                return;
        postOrder(edgeList[node][0]);
        postOrder(edgeList[node][1]);
        print(node, end="");
n = int(sys.stdin.readline().rstrip());
edgeList = {};
for i in range(n):
        parent, left, right = sys.stdin.readline().split();
        edgeList[parent] = [None, None];
        if left == ".":
                pass;
        else:
                edgeList[parent][0] = left;
        if right == ".":
                pass;
        else:
                edgeList[parent][1] = right;
preOrder("A");
print();
inOrder("A");
print();
postOrder("A");
print();

```

연습문제 22.

```
import sys

# 인접리스트 생성
def makeList(n):
        lst = [[] for _ in range(n+1)]
        for i in range(n-1):
                tmp = list(map(int, sys.stdin.readline().split()))
                lst[tmp[0]].append(tmp[1])
                lst[tmp[1]].append(tmp[0])
                return lst

# 인접리스트 1부터 돌면서 훑으면서 부모 노드 계산하기
def DFS(i):
        global visitedAry, edgeList;
        stack = [];
        stack.append(i);
        visitedAry[i] = True;
        while len(stack)>0:
                tmp = stack.pop();
                for v in edgeList[tmp]:
                        # print(v);
                        if visitedAry[v]== False:
                                stack.append(v);
                                visitedAry[v] = tmp;

def printAns(ans):
        for i in range(2, len(ans)):
                print(ans[i]);

n = int(sys.stdin.readline().rstrip())
edgeList = makeList(n)

visitedAry = [False for _ in range(n+1)];

DFS(1);
printAns(visitedAry);
```

연습문제 23-1.

```
import sys;
from collections import deque;
from collections import deque

def printEdgeList(edgeList):
        for i in range(1, len(edgeList)):
                print(f"{i} --> {edgeList[i]}");
def DFS(i):
        global edgeList;
        # visitedAry = [False for _ in range(node+1)];
        seq = [];
        stack = [];
        stack.append(i);
        while len(stack)>0:
                tmp = stack.pop();
                # print(tmp, end=" ");
                if tmp not in seq:
                        seq.append(tmp);
                        for v in edgeList[tmp]:
                                # print(v);
                                # if v not in seq:
                                stack.append(v);
                                # visitedAry[v] = True;
        for i in seq:
                print(i, end= " ");
        print();

def BFS(i):
        global edgeList;
        seq = [];
        dq = deque();
        dq.append(i);
        while len(dq)>0:
                tmp = dq.popleft();
                if tmp not in seq:
                        seq.append(tmp);
                        for v in edgeList[tmp]:
                                dq.append(v);
        for i in seq:
        print(i, end =" ");
```

연습문제 23-2.

```
41  node, edge, start = map(int, sys.stdin.readline().split());
42
43  # visitedAry = [False for _ in range(node+1)];
44  edgeList = [[] for _ in range(node +1)];
45
46  for i in range(edge):
47          a, b = map(int, sys.stdin.readline().split());
48          edgeList[a].append(b);
49          edgeList[b].append(a);
50
51  for i in edgeList:
52          i.sort(reverse=True);
53
54  DFS(start);
55
56  for i in edgeList:
57          i.sort(reverse = False);
58  # printEdgeList(edgeList);
59  BFS(start)
```

연습문제 24.

```
import sys
sys.setrecursionlimit(10000)

def printEdge(edgeList):
        for i in range(1, len(edgeList)):
                print(f"{i} --> {edgeList[i]}")

def DFS(i):
        global edgeList, visitedAry, count;
        count+=1;
        visitedAry[i] = True;
        for v in edgeList[i]:
                if not visitedAry[v]:
                        DFS(v);


node = int(sys.stdin.readline().rstrip());
edge = int(sys.stdin.readline().rstrip());
count = 0;
edgeList = [[] for _ in range(node+1)]
visitedAry = [False for _ in range(node+1)]


for i in range(edge):
        a, b = map(int, sys.stdin.readline().split());
        edgeList[a].append(b);
        edgeList[b].append(a);
# printEdge(edgeList);
DFS(1);
print(count-1);
```

연습문제 25.

```
import sys;
sys.setrecursionlimit(10000)

def printEdgeList(edgeList):
        for i in range(1, len(edgeList)):
                print(f"{i} --> {edgeList[i]}")

def printVisitedAry(visitedAry):
        for i in range(1, len(visitedAry)):
                print(f"{visitedAry[i]}", end= " ");
        print();

def DFS(i):
        global visitedAry, edgeList;
        visitedAry[i] = True;
        for v in edgeList[i]:
                if not visitedAry[v]:
                        DFS(v);

node, edge = map(int, sys.stdin.readline().split());
edgeList = [[] for _ in range(node+1)];

for i in range(edge):
        a, b = map(int, sys.stdin.readline().split());
        edgeList[a].append(b);
        edgeList[b].append(a);
visitedAry = [False for _ in range(node+1)]

count = 0;
for i in range(1, node+1):
        if visitedAry[i] == False:
        # print(i)
        count+=1;
        DFS(i);

print(count);
# printEdgeList(edgeList);
# printVisitedAry(visitedAry);
```

연습문제 26.

```
import sys;

node, edge = map(int, sys.stdin.readline().split());
start_node = int(sys.stdin.readline().rstrip());
edgeList = [[] for _ in range(node+1)]

for i in range(edge):
        start, end, weight = map(int, sys.stdin.readline().split());
        edgeList[start].append((end, weight));

INFINITE = sys.maxsize;

distance = [INFINITE for _ in range(node+1)];
visited = [False for _ in range(node +1)];
visited[0] = True;

from heapq import *;

h = [];
heappush(h, (0, start_node));
distance[start_node] = 0;

while len(h) > 0:
        now = heappop(h);
        now_node = now[1];
        now_value = now[0]
        if visited[now_node]:
                continue;
        visited[now_node] = True;
        for next, value in edgeList[now_node]:
                distance[next] = min(distance[next], distance[now_node] +
                value);
                heappush(h, (distance[next], next))


for i in range(1, node+1):
        if visited[i]:
                print(distance[i]);
        else:
        print("INF");
```

연습문제 27-1.

```
import sys;
from collections import deque;

def printEdge(edgeList):
        for i in range(1, city+1):
                print(f"{i} --> {edgeList[i]}")
        print();

def printInfo(lst):
        for i in range(1, city+1):
                print(lst[i], end=" ");
        print();


city = int(sys.stdin.readline().rstrip());
road = int(sys.stdin.readline().rstrip());

indegree = [0 for _ in range(city+1)];
time = [0 for _ in range(city + 1)];

edgeList = [[] for _ in range(city+1)]
reverseEdgeList = [[] for _ in range(city+1)]

for i in range(road):
        a, b, c = map(int, sys.stdin.readline().split());
        edgeList[a].append((b, c));
        reverseEdgeList[b].append((a, c));
        indegree[b] +=1;


start, end = map(int, sys.stdin.readline().split());
queue = deque();
queue.append(start);
```

연습문제 27-2.

```
while queue:
        now = queue.popleft();
        for next in edgeList[now]:
                next_index = next[0];
                next_time = next[1];
                indegree[next_index]-=1;
                if indegree[next_index] ==0:
                        queue.append(next_index);
                        time[next_index] = max(time[next_index], next_time
                        + time[now])


# printEdge(edgeList);
# printEdge(reverseEdgeList)
# printInfo(indegree)
# printInfo(time);

print(time[-1])
count = 0;

queue.clear();
visited = [False]*(city+1);

queue.append(end);
visited[end] = True;

while queue:
        now = queue.popleft();
        for next in reverseEdgeList[now]:
                if time[now] - time[next[0]] == next[1]:
                        count+=1;
                        if visited[next[0]] == False:
                                queue.append(next[0]);
                                visited[next[0]] = True;
print(count)
```

연습문제 28-1.

```
import sys;
from collections import deque;


def BFS():
        dx = [0, 0, -1, 1];
        dy = [1, -1, 0, 0];

        global row, col, time, box, visited
        while len(queue)>0:
                now = queue.popleft();
                now_x = now[0];
                now_y = now[1];
                visited[now_x][now_y] = True;
                for w in range(4):
                        tmp_x = now_x + dx[w];
                        tmp_y = now_y + dy[w];
                                if box[tmp_x][tmp_y] == 0 and
                                visited[tmp_x][tmp_y] == False:
                                        queue.append((tmp_x, tmp_y))
                                        visited[tmp_x][tmp_y] = True;

                                if time[tmp_x][tmp_y] == 0:
                                        time[tmp_x][tmp_y] =
                                        time[now_x][now_y]+1;
                                else:
                                        time[tmp_x][tmp_y] =
                                        min(time[now_x][now_y] + 1,
                                        time[tmp_x][tmp_y]);
```

연습문제 28-2.

```
col, row = map(int, sys.stdin.readline().split());

box = [[-1 for _ in range(col+2)] for _ in range(row+2)];
visited = [[False for _ in range(col+2)] for _ in range(row+2)]
time = [[0 for _ in range(col+2)] for _ in range(row+2)]
for i in range(1, row+1):
        tmp = list(map(int, sys.stdin.readline().split()));
        for j in range(1, len(tmp)+1):
                box[i][j] = tmp[j-1]

queue = deque();
for i in range(1, row+1):
        for j in range(1, col+1):
                if box[i][j] == 1:
                        queue.append((i, j));

BFS()

result = max(map(max, time))
for i in range(1, row+1):
        for j in range(1, col+1):
                if time[i][j] == 0 and box[i][j]==0:
                        result = -1;
                        break;


print(result);
```

연습문제 29-1.

```
import sys;
sys.setrecursionlimit(10**6)

class TreeNode():
        def __init__(self):
                self.data = None;
                self.left = None;
                self.right = None;



def postOrder(current):
        if current == None:
                return;
        postOrder(current.left);
        postOrder(current.right);
        print(current.data);



n = int(sys.stdin.readline().rstrip());

node = TreeNode();
node.data = n;
root = node;
lst = []
```

연습문제 29-2.

```
while True:
        try:
                lst.append(int(sys.stdin.readline()));
        except:
                break;


for k in lst:
        node = TreeNode();
        node.data = k;
        current = root;
        while True:
                if k > current.data:
                        if current.right ==None:
                                current.right = node;
                                break;
                        else:
                                current = current.right;
                else:
                        if current.left ==None:
                                current.left = node;
                                break;
                        else:
                                current = current.left;

postOrder(root);
```

연습문제 30.

```
import sys;
from heapq import *;

city = int(sys.stdin.readline().rstrip());
bus = int(sys.stdin.readline().rstrip());
edgeList = [[] for i in range(city+1)];

for i in range(bus):
        start, end, weight = map(int, sys.stdin.readline().split());
        edgeList[start].append((end, weight));

start, end = map(int, sys.stdin.readline().split());
visited = [False for _ in range(city+1)];
visited[0] = True;
max_value = 2100000000;
lst = [max_value for _ in range(city+1)]
lst[start] = 0;

q = [];
heappush(q, (0, start))
# visited[start] = True;

while len(q)>0:
        value, now = heappop(q);
        # print(value, now)


if visited[now] == False:
        visited[now] = True;
        for next, next_weight in edgeList[now]:
                lst[next] =min(next_weight + lst[now], lst[next]);

                heappush(q, (lst[next], next))
print(lst[end])
# print(lst)
```