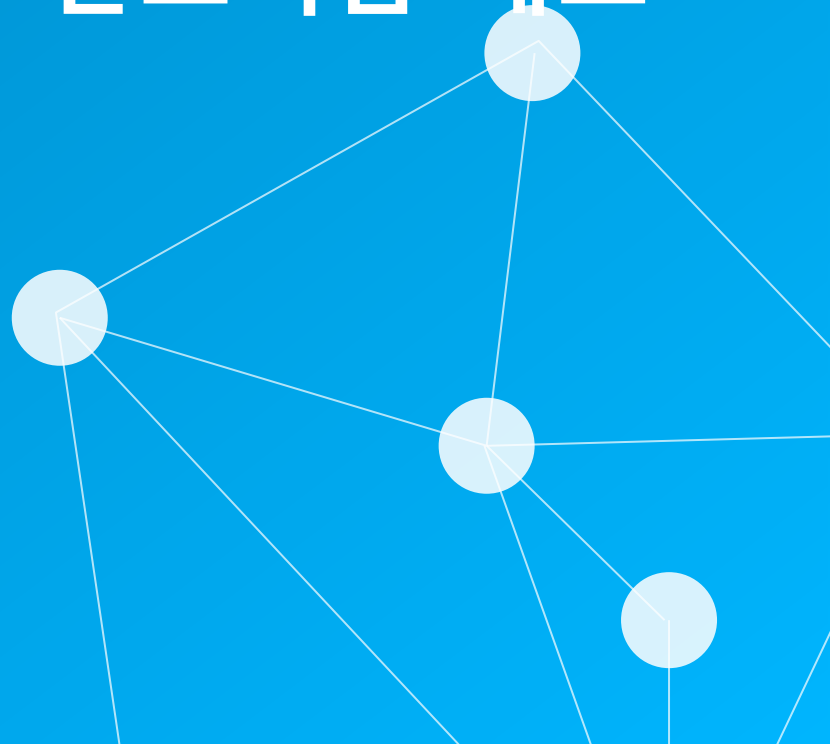


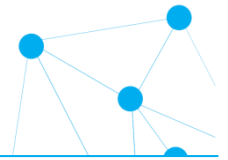
파이썬 알고리즘

01 CHAPTER

알고리즘 개요



학습 내용



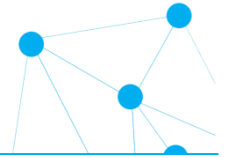
1.1 알고리즘이란?

1.2 문제 해결 과정

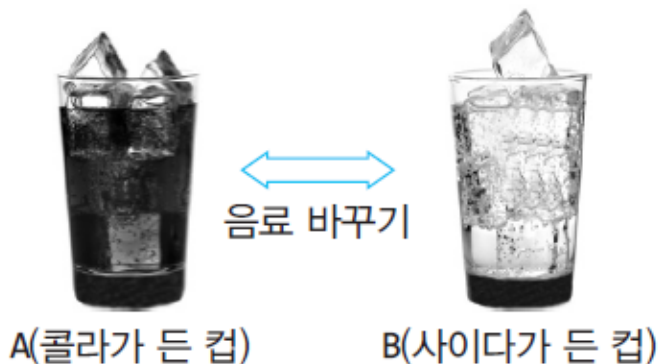
1.3 중요한 문제의 유형들

1.4 기본적인 자료구조와 파이썬

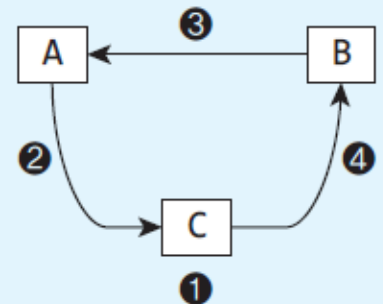
1.1 알고리즘이란?



- 알고리즘(algorithm)
 - 해결해야 할 어떤 문제가 주어졌을 때, 이 문제의 해답을 구하기 위한 절차를 순서대로 명확하게 나타낸 것
 - 아부 압둘라 무함마드 이븐 무사 알-콰리즈미(al-Khwarizmi)
- 예) 두 컵의 음료 바꾸기



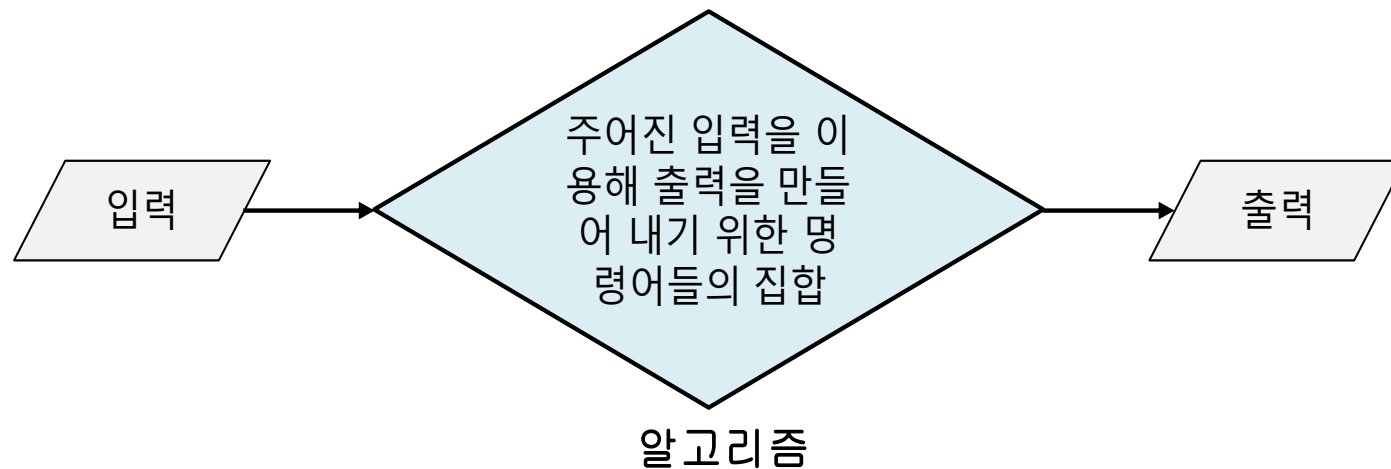
- ① 새로운 컵 C를 준비한다.
- ② A의 콜라를 모두 C에 붓는다.
- ③ B의 사이다를 A에 모두 붓는다.
- ④ C의 내용물을 B에 모두 붓는다.



알고리즘의 정의



- 주어진 문제를 해결하기 위한 단계적인 절차
 - 알고리즘은 C언어나 Java, 파이썬 등과 같은 프로그래밍 언어와 상관없이 **문제 해결 절차를 나타내는 명령어의 집합**

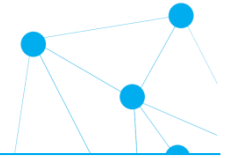


알고리즘의 조건



- 입력(well-defined inputs): 알고리즘이 입력을 받는다면 모호하지 않고 잘 정의된 입력이어야 한다. 알고리즘은 0개 이상의 입력을 갖는다.
- 출력(well-defined outputs): 출력은 명확하게 정의되어야 하며 1개 이상의 출력이 반드시 존재하여야 한다.
- 명확성(clear and unambiguous): 각 명령어의 의미는 모호하지 않고 명확해야 한다.
- 유한성(Finite-ness): 한정된 수의 단계 후에는 반드시 종료되어야 한다. 즉, 무한루프나 이와 유사한 상태로 끝나서는 안 된다.
- 유효성(Feasible): 명령어들은 현재 실행 가능한 연산이어야 한다. 미래에 개발될 기술 등을 포함해서는 안 된다.

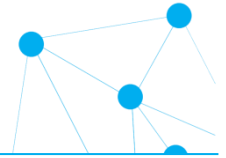
알고리즘의 기술: 자연어, 흐름도



① 영어나 한국어와 같은 자연어를 사용하는 방법

② 흐름도(flowchart)로 표시하는 방법

알고리즘의 기술: 유사코드, C언어



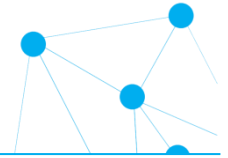
③ 유사 코드(pseudo-code)로 기술하는 방법

자연어보다는 체계적
프로그래밍언어보다는 덜 엄격

④ 특정한 프로그래밍 언어(예: C언어)

C언어와 같은 프로그래밍 언어는 바로 실행 확인
문법을 정확히 구현 불필요한 표현들 표현

알고리즘의 기술: 파이썬



- 유사 코드 표현과 매우 유사함

```
01 def find_max( A ):
02     max = A[0]
03     for i in range(len(A)) :
04         if A[i] > max :
05             max = A[i]
06     return max
```

```
01 find_max( A )
02     max←A[0]
03     for i←1 to size(A) do
04         if A[i] > max then
05             max←A[i]
06     return max
```

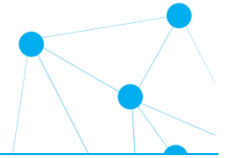
- 바로 실행하여 결과를 확인할 수도 있다!

알고리즘 테스트 최댓값 찾기

```
array = [ 20, 34, 12, 93, 84, 39, 64, 55, 24 ]
print(array, "최댓값=", find_max(array))
```

```
C:\WINDOWS\system32\cmd.exe
[20, 34, 12, 93, 84, 39, 64, 55, 24] 최댓값= 93
```


최대 공약수 문제



두 자연수 a 와 b 의 최대 공약수(greatest common divisor)를 구하라. a 와 b 의 최대 공약수는 a 의 약수인 동시에 b 의 약수인 숫자 중에서 가장 큰 수를 의미한다.

- 알고리즘 1: 정의를 직접 이용

$\text{gcd}(a, b)$

1. a 의 약수를 모두 찾아 리스트 alist 에 저장한다.
2. b 의 약수를 모두 찾아 리스트 blist 에 저장한다.
3. alist 와 blist 에 공통적으로 들어 있는 가장 큰 숫자를 찾아 반환한다.

- 알고리즘 2: 한 수의 약수만 구함



60

`gcd(a, b)`

1. a의 약수를 모두 찾아 리스트 `alist`에 저장한다.
2. `alist`의 가장 큰 수부터 차례대로 b의 약수인지를 검사한다. 만약 b의 약수이기도 하면 이 숫자를 반환한다.
3. 이 과정을 `alist`의 모든 숫자에 대해 반복한다.

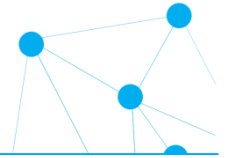
- 알고리즘 3: 유클리드 알고리즘

$$\gcd(a, b) = \gcd(b, a \bmod b)$$

– 예)

```
01 def gcd(a, b) :           # a가 b보다 작지 않아야 함
02     while b != 0 :         # b가 0이 아닐 때까지
03         r = a % b
04         a = b
05         b = r
06     return a               # 결과 반환
```

알고리즘은...

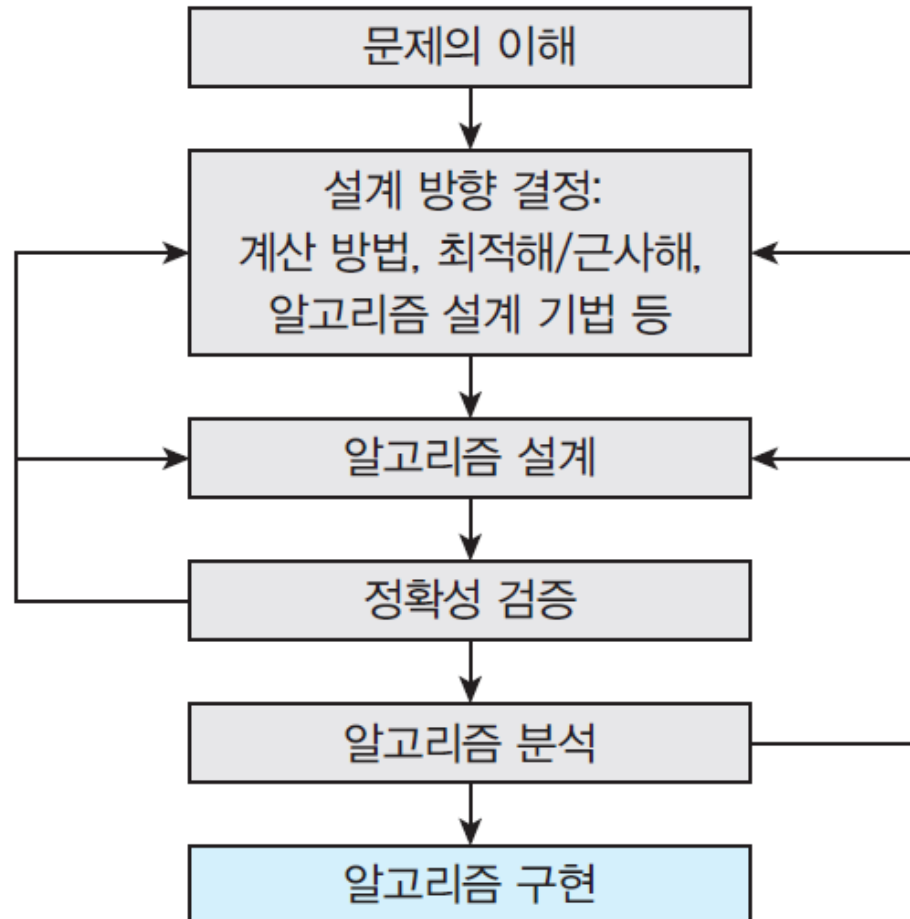


- 알고리즘은 여러 가지 방법으로 표현할 수 있다.
- 하나의 문제를 해결하기 위해 여러 가지 알고리즘이 가능하다.
- 동일한 문제에 대해 매우 다른 전략을 기반으로 알고리즘을 작성할 수 있고, 이들은 매우 다른 속도로 문제를 해결할 수 있다.
- 알고리즘의 정확한 동작을 위한 제한 조건이나 입력의 범위를 신중하게 고려해야 한다.
 - 예) 유클리드 알고리즘에서는 a 가 b 보다 커야 한다.

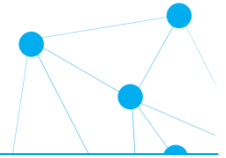
1.2 문제 해결 과정



- 알고리즘 개발 과정

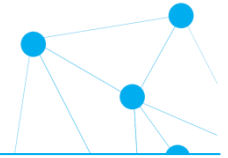


문제의 이해



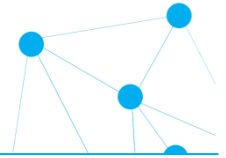
- 문제를 정확히 이해하는 것이 가장 중요
 - 간단한 입력 예에 대한 해를 구해보고,
 - 좀 더 특별한 경우에 대해서도 생각
- 입력(input)은 주어진 문제의 하나의 사례(instance)
 - 입력의 범위
 - 올바른 알고리즘은 “대부분의 입력”이 아니라 “모든 유효한 입력”에 대해 정확한 해답을 구함

개발 방향 결정과 알고리즘의 설계



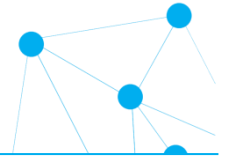
- 알고리즘 설계 전 결정해야 할 사항들
 - 순서적(sequential) 알고리즘 / 병렬처리(parallel) 알고리즘
 - 최적해와 근사해
- 근사 알고리즘(10장)을 고려해야 하는 상황
 - 많은 사례에 대해 정확한 해를 구할 수 없는 경우: 2의 제곱근 등
 - 계산량이 너무 많아 현실적인 시간이 불가능한 경우: 10장
 - 알고리즘의 중간 단계에서 사용되는 경우: 9장의 분기 한정 등

알고리즘의 정확성



- 실험적 분석(Experimental analysis, testing)
 - 다양한 입력 적용
 - 충분한 테스트가 어느 정도인지 애매함
 - 알고리즘이 틀렸다는 것을 보여주기 위해서는 **한 가지 입력** 사례만으로 충분
- 증명적인 분석(Formal analysis, proving)
 - 수학적 증명
 - 수학적 귀납법(mathematical induction) 등
 - 증명이 매우 어려울 수도 있음

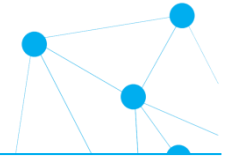
알고리즘의 분석과 구현



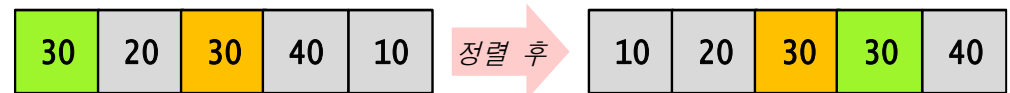
- 알고리즘의 효율성 분석
 - 시간 효율성
 - 공간 효율성
 - 코드 효율성
- 알고리즘 구현
 - 특정 프로그래밍 언어
 - 컴파일 / 인터프리터

알고리즘은 흔히 '**시간과 공간이 트레이드 오프**' 관계
실행 시간이 빠른 알고리즘은 공간을 많이 사용,
공간을 적게 차지하는 알고리즘은 실행 시간이 느리다.

1.3 중요한 문제의 유형들



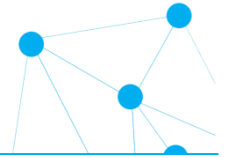
- 정렬(3~6장)
 - 데이터를 순서대로 재배열하는 작업
 - 오름차순 / 내림차순
 - 레코드, 정렬 키(key)
 - 비교기반 / 분배기반(기수 정렬)
 - 안정성 만족 / 불만족
 - 제자리 정렬



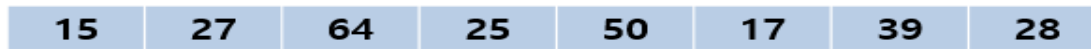
상대적인 위치가 바뀜 → 안정성을 충족하지 않음

- 탐색(정렬은 탐색에서 매우 중요)
 - 원하는 값을 가진 레코드를 찾는 작업
 - 탐색키
 - 순차 탐색(3장), 이진 탐색(4장), 해싱(6장)

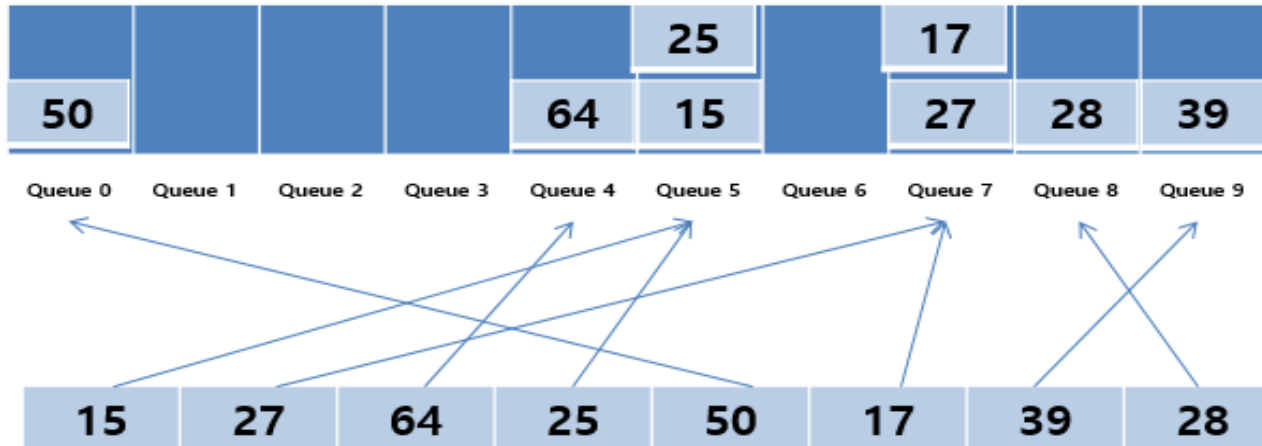
기수 정렬



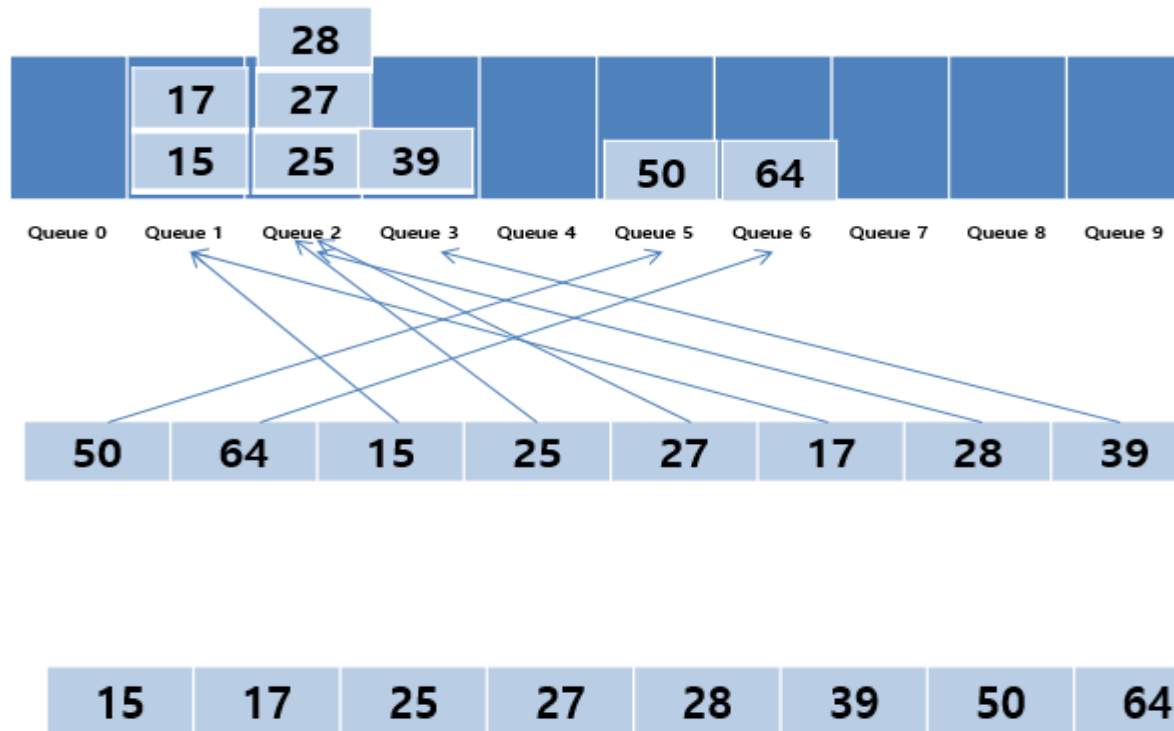
- 기수정렬은 낮은 자리수부터 비교하여 정렬
- 비교 연산X
- 정렬 속도가 빠르지만 데이터 전체 크기에 기수 테이블의 크기만한 메모리가 더 필요
 1. 0~9 까지의 Bucket(Queue 자료구조의)을 준비한다.
 2. 모든 데이터에 대하여 가장 낮은 자리수에 해당하는 Bucket에 차례대로 데이터를 둔다.
 3. 0부터 차례대로 버킷에서 데이터를 다시 가져온다.
 4. 가장 높은 자리수를 기준으로 하여 자리수를 높여가며 2번 3번 과정을 반복한다.



1의 자리에 해당되는 Queue에 데이터를 위치



10의 자리에 해당되는 Queue에 데이터를 위치



- 문자열 처리

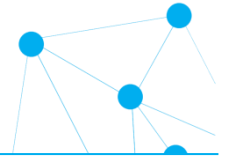
- 문자열(string)은 문자들의 시퀀스(sequence)
- 텍스트 문자열 / 비트 문자열(bit string) / 유전 시퀀스 등
- 문자열 매칭(string matching) 문제: 3장, 6장

- 그래프 문제

- 연결된 객체들 사이의 관계를 표현할 수 있는 자료구조
- 다양한 객체(정점)들이 서로 복잡하게 연결(간선)된 구조 표현
- 다양한 문제들
 - 순회 (3장)
 - 위상 정렬(4장)
 - 최단경로(7,8장)
 - 최소비용의 신장트리 (8장)
 - TSP, Graph Coloring(9장) 등

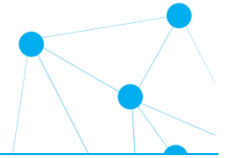
- 조합 문제(Combinatorial problems)
 - 어떤 조건을 만족하는 순열이나 조합 또는 부분 집합과 같은 **조합 객체(combinatorial object)**를 찾는 문제
 - 예: 외판원 TSP 문제(모든도시를 정확히 한번 방문하는 최단경로를 찾는 문제)
 - 모든 가능한 경로(조합 객체) 중에서 최단 경로란 추가적인 특성을 갖는 조합 객체를 찾는 문제
 - 보통 조합 객체의 수가 문제의 크기에 따라 매우 빠르게 증가
 - 컴퓨팅에서 가장 어려운 문제: 10장
- 기하학적 문제
 - 고대의 그리스인들: 자와 컴퍼스
 - 최근접 쌍의 거리 문제(closest-pair problem): 3장, 5장
 - 컨벡스 헐(convex-hull)
 - 계산 기하학

1.4 기본적인 자료구조와 파이썬



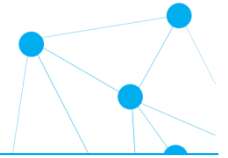
- 자료구조
 - 자료들을 정리하고 조직화하는 여러 가지 구조
 - 알고리즘의 설계에 큰 영향을 미침
 - 단순 자료구조 / 복합 자료구조
 - 배열 구조와 연결된 구조
 - 직접 접근 / 순서 접근
- 파이썬에서의 배열: 리스트(list)와 튜플(tuple)로 구현 가능

리스트



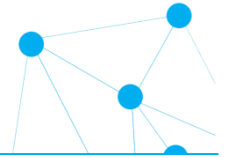
- 항목들이 순서대로 나열, 각 항목들은 위치를 갖음
- 파이썬에서 자료구조 "리스트"가 필요하면
 - 파이썬의 리스트를 사용
 - 파이썬 리스트
 - 자료구조 리스트를 배열 구조로 구현한 클래스
 - 가장 많이 사용됨

스택



- **후입선출(LIFO: Last-In First-Out)**
- 파이썬에서 스택이 필요하다면
 - 방법 1: 파이썬 리스트 이용
 - 방법 2: 큐 모듈(queue)의 LifoQueue 클래스 사용
 - 방법 3: 직접 클래스로 구현해서 사용

큐



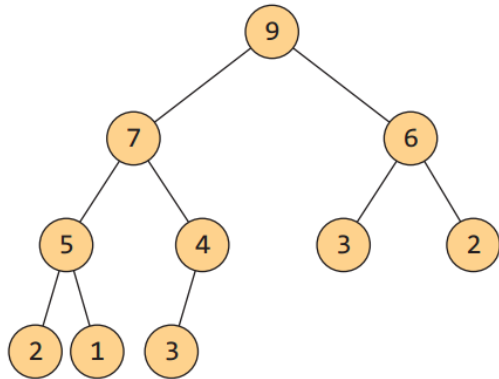
- **선입선출(FIFO: First-In First-Out)**
- 파이썬에서 큐가 필요하다면
 - 방법 1: 큐 모듈(queue)의 Queue 클래스 사용
 - 방법 2: 직접 클래스로 구현해서 사용

우선순위 큐

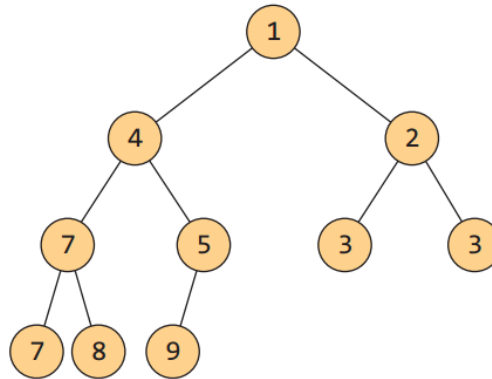


- 우선순위의 개념을 큐에 도입한 자료구조
 - 선형 자료구조가 아님
 - 힙(heap)이 가장 효율적인 구현 방법

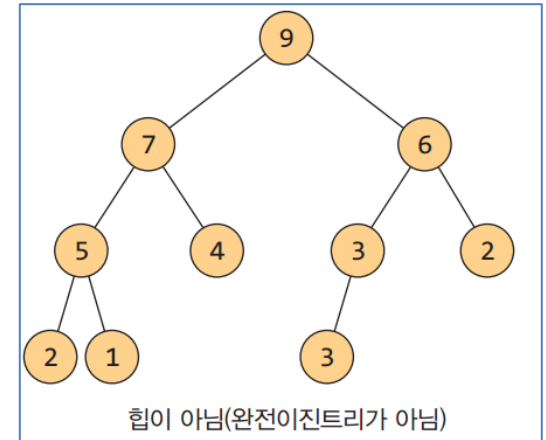
- 파이썬에서 우선순위 큐가 필요하면
 - heapq 모듈을 사용



최대 힙(Max Heap)

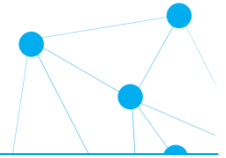


최소 힙(Min Heap)



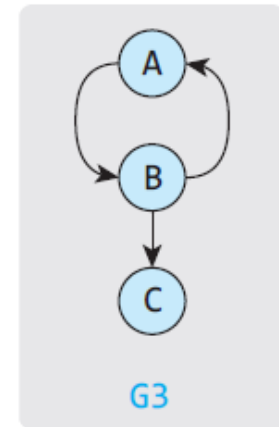
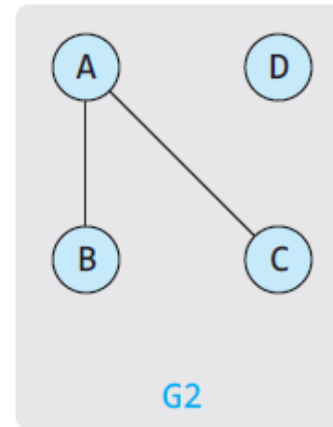
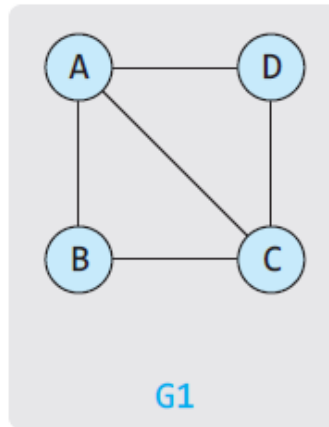
힙이 아님(완전이진트리가 아님)

그래프

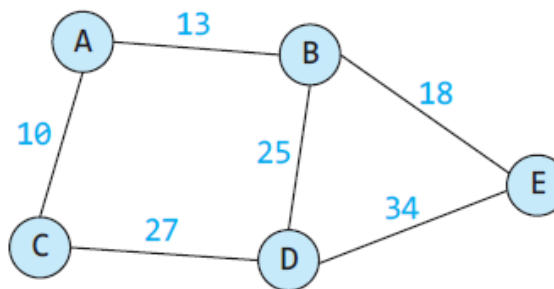


- $G = (V, E)$
 - V: 정점. 객체(object)를 표현
 - E: 간선. 객체들 사이의 관계

- 그래프의 종류
 - 방향 그래프
 - 무방향 그래프



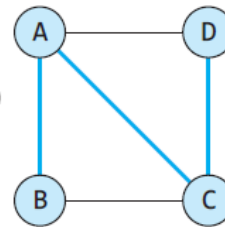
- 가중치 그래프



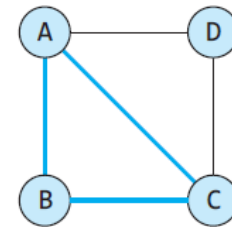
- 그래프 용어

- 인접 정점(adjacent vertex)
- 정점의 차수(degree)
 - 진입 차수 / 진출 차수
- 경로(path)
- 경로의 길이
- 단순 경로/사이클(cycle)
- 연결 그래프
- 트리
- 완전 그래프

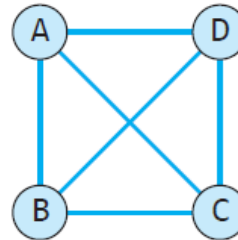
경로 B, A, C, D는
단순 경로



경로 B, A, C, B는
사이클



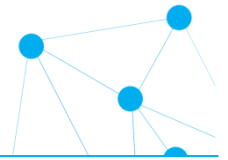
모든 정점 간에
간선이 존재하니까
완전 그래프!



- 그래프의 표현

- 인접 행렬 표현: 7.5절, 9.4절 등
- 인접 리스트 표현: 3.6절, 4.2절

트리



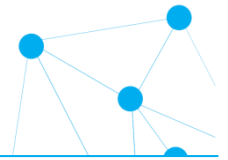
- **자유트리(free tree)**
 - 사이클이 없는 연결 그래프(connected acyclic graph)
- **루트를 가진(rooted) 트리**
 - 자유트리의 정점들 중에 하나를 루트(root)로 선택

- 트리 용어

- 트리의 표현

- 일반 트리
- 이진 트리: 5.4절
 - 배열 구조 / 연결된 구조
- 트리의 개념: 3, 8, 9, 10장

집합



- 원소들 사이에 순서가 없고, 중복을 허용하지 않음
 - 선형 자료구조가 아님
 - "위치"가 없음

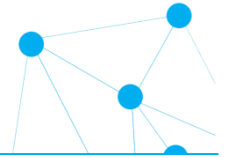
- 파이썬의 집합

```
s1 = { 1,2,3 }           # 집합 객체
s2 = { 2,3,4,5 }         # 집합 객체
s3 = s1.union(s2)         # 합집합
s4 = s1.intersection(s2)  # 교집합
s5 = s1 - s2              # 차집합
```

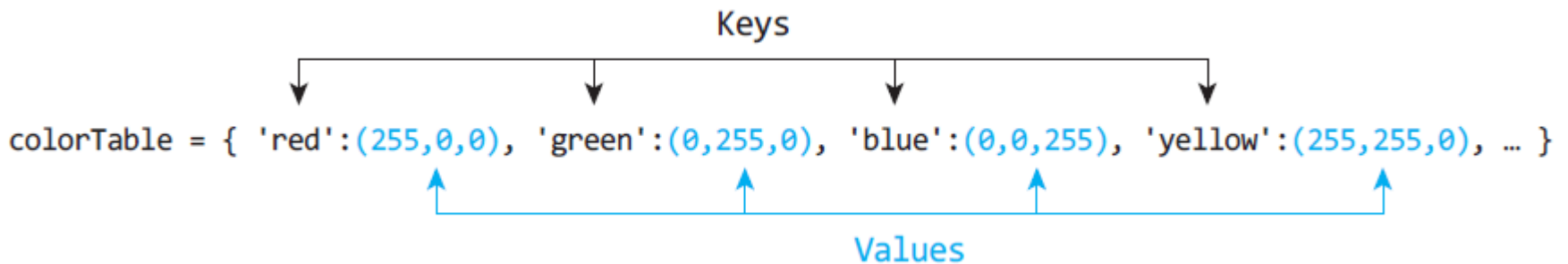
```
print("s1:", s1)
print("s2:", s2)
print("s3:", s3)
print("s4:", s4)
print("s5:", s5)
```

```
s1: {1, 2, 3}
s2: {2, 3, 4, 5}
s3: {1, 2, 3, 4, 5}
s4: {2, 3}
s5: {1}
```

맵 또는 딕셔너리

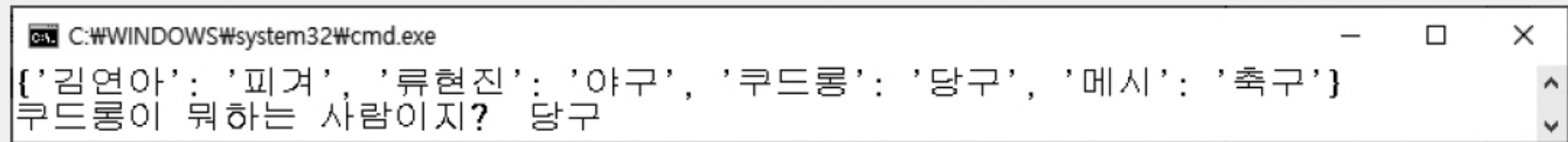


- 탐색을 위한 자료구조
- 키를 가진 레코드 또는 엔트리(entry)의 집합
- 엔트리
 - 키(key): 영어 단어와 같은 레코드를 구분할 수 있는 탐색키
 - 값(value): 영어 단어의 의미와 같이 탐색키와 관련된 정보들
- 파이썬의 딕셔너리



• 딕셔너리 사용 예

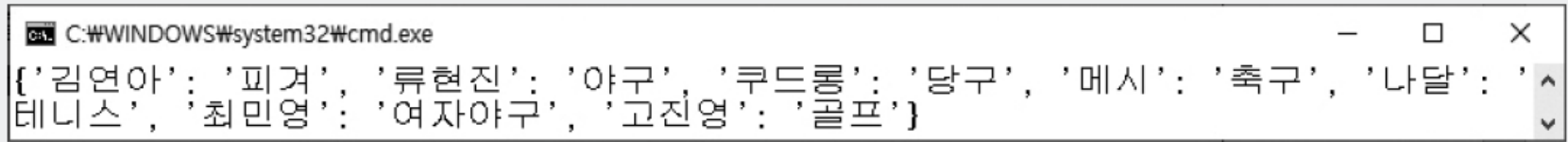
```
map = { '김연아': '피겨', '류현진': '야구', '쿠드롱': '당구', '메시': '축구' }  
print(map)  
print('쿠드롱이 뭐하는 사람이지? ', map['쿠드롱'])
```



C:\WINDOWS\system32\cmd.exe

```
{ '김연아': '피겨', '류현진': '야구', '쿠드롱': '당구', '메시': '축구' }  
쿠드롱이 뭐하는 사람이지? 당구
```

```
map['나달'] = '테니스' # 맵에 하나의 항목 추가(항목 변경 코드가 아님)  
map.update({'최민영': '여자야구', '고진영': '골프'}) # 여러 항목 추가  
print(map)
```

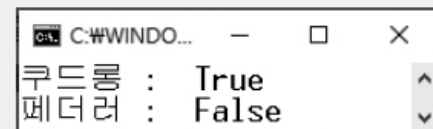


C:\WINDOWS\system32\cmd.exe

```
{ '김연아': '피겨', '류현진': '야구', '쿠드롱': '당구', '메시': '축구', '나달': '테니스', '최민영': '여자야구', '고진영': '골프' }
```

in 연산자를 이용하면 어떤 키가 딕셔너리에 있는지를 검사할 수 있다.

```
print('쿠드롱 : ', '쿠드롱' in map)  
print('페더러 : ', '페더러' in map)
```

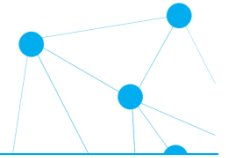


C:\WINDO...

```
쿠드롱 : True  
페더러 : False
```

실습 과제





감사합니다!