자료구조의 이해

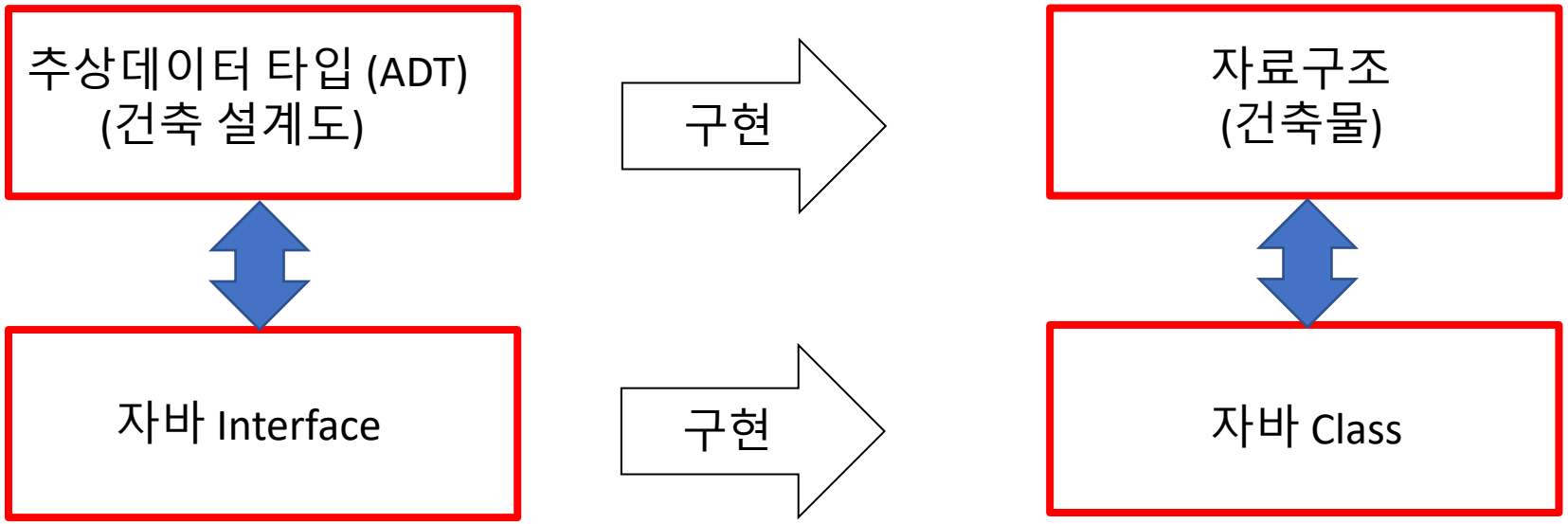
Chapter 01 ~ 02

자료구조를 배우기 위한 준비

1.1 자료구조와 추상데이터타입

□ 자료구조란?

- 일련의 동일한 타입의 데이터를 정돈하여 저장한 구성체를 의미
- 탐색, 삽입, 삭제 등의 연산을 효율적으로 수행하기 위해
- 추상데이터타입을 구체적으로 구현한 것을 의미
- 인터페이스를 클래스로 구현한 것을 자료구조로 볼 수 있다.



1.2 수행시간의 분석

- 자료구조의 효율성은 수행되는 연산의 수행시간으로 측정

- **성능** :
  - **시간 복잡도** : 수행시간을 나타냄
  - **공간 복잡도** : 메모리 공간의 크기를 나타냄

□ **시간 복잡도** : 알고리즘이 실행되는 동안에 사용된 기본적인 연산 횟수를 입력 크기의 함수로 나타냄

- 최악경우 분석
- 평균경우 분석
- 최선경우 분석
- 상각분석

1.3 수행시간의 점근표기법

☐ O (Big-Oh) 표기법

- 알고리즘의 성능을 수학적으로 표현해주는 표기법
- 시간과 공간의 복잡도를 표현
- 실제 러닝타임의 표기보다는 **데이터나 사용자의 증가율에 따른 알고리즘의 성능 예측**이 목표

※ 자주 사용되는 O표기

- $O(1)$
- $O(N)$
- $O(N^2)$
- $O(N^3)$
- $O(2^N)$
- O(logN

1.4 순환

□ 순환

- 메소드의 실행 과정 중 스스로를 호출하는 것.
- 팩토리얼, 트리 자료구조, 프랙털 등

```java
public class Recursion{
    public void recurse() {
        System.out.println("*");
        recurse();
    }

    public static void main(String[] args) {
        Recursion r = new Recursion();
        r.recurse();
    }
}
```

```java
public class Recursion{
    public void recurse(int count) {
        if (count <=0)
            System.out.println(".");
        else {
            System.out.println(count+"*");
            recurse(count - 1);
        }
    }
    public static void main(String[] args) {
        Recursion r = new Recursion();
        r.recurse(5);
    }
}
```

※ 무한 호출을 방지해야 함

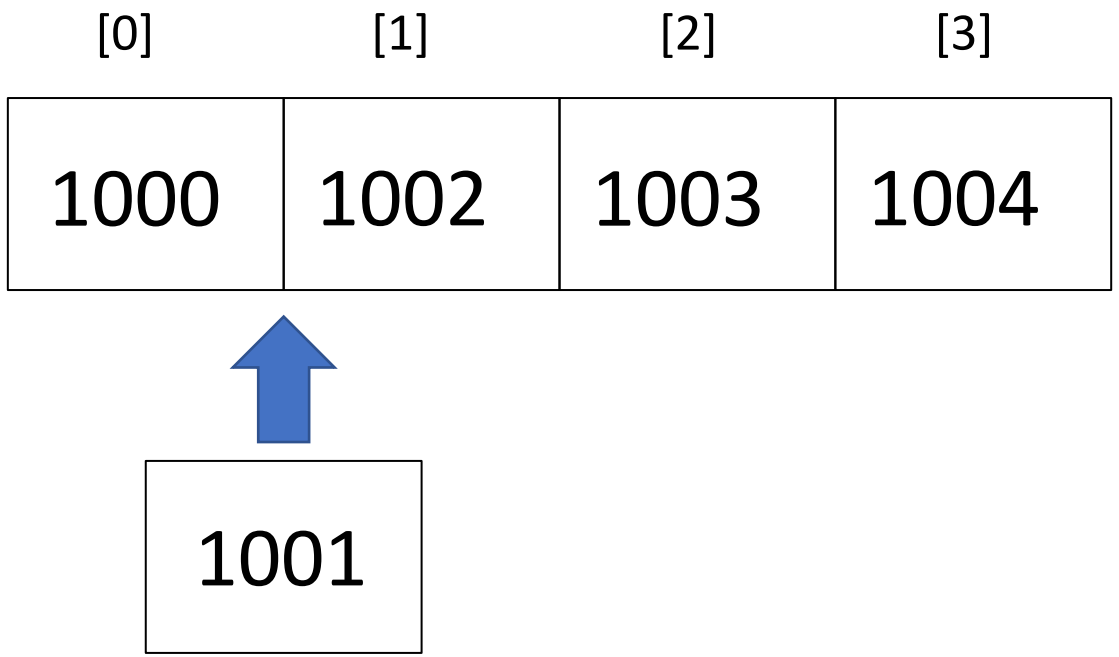
리스트

2.1 배열

□ 배열

- 동일한 타입의 원소들이 연속적인 메모리 공간에 할당되어
각 항목이 하나의 원소에 저장되는 자료구조

- 삽입, 삭제와 같은 변동 발생 시 많은 시간이 걸림.



2.1 배열 수행시간

□ 탐색 : 인덱스로 배열 원소를 직접 접근, O(1) 시간에 수행 가능

- 탐색에서 빠른 성능을 기대할 수 있다.

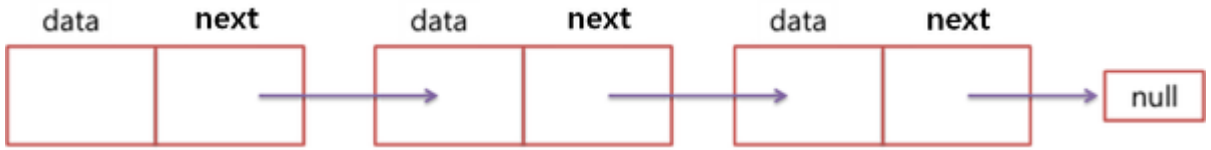
□ 삭제, 삽입 : 최악의 경우 O(N) 시간이 소요.

- 새 항목을 가장 앞에 삽입하거나 첫 항목을 삭제할 시 최악의 경우.
- 자료의 삽입과 삭제에 **비효율적.**
- 그러나 상각분석에 따르면 삽입이나 삭제의 평균 수행 시간은 **O(1)** 이다.

2.2 단순연결리스트

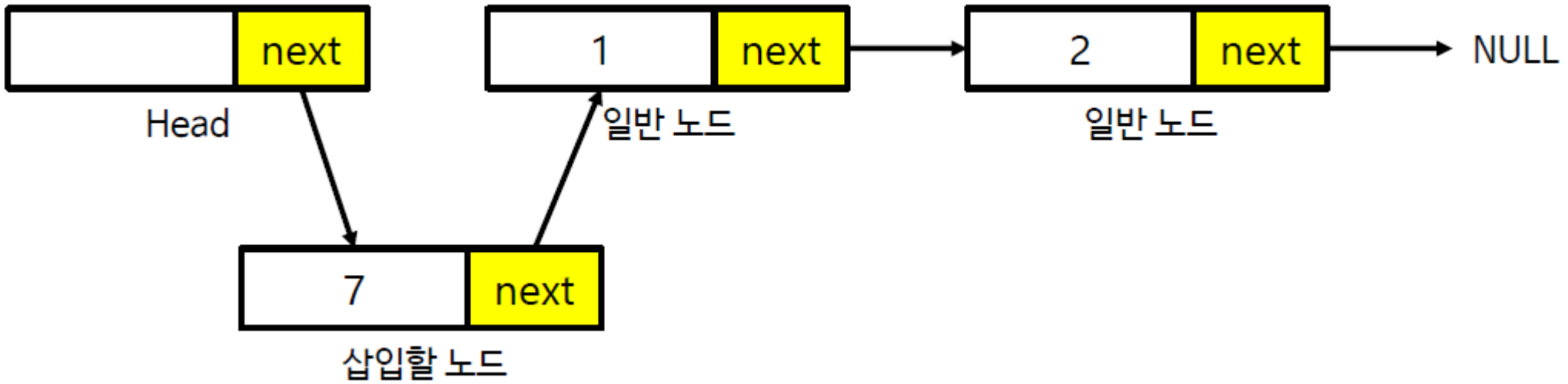
□ 단순연결리스트

- 동적 메모리 할당을 받아 노드를 저장
- 노드는 레퍼런스를 이용하여 다음 노드를 가리키도록 만듦
- 노드들을 한 줄로 연결시킨 것.
- 순차탐색 (Sequential Search)

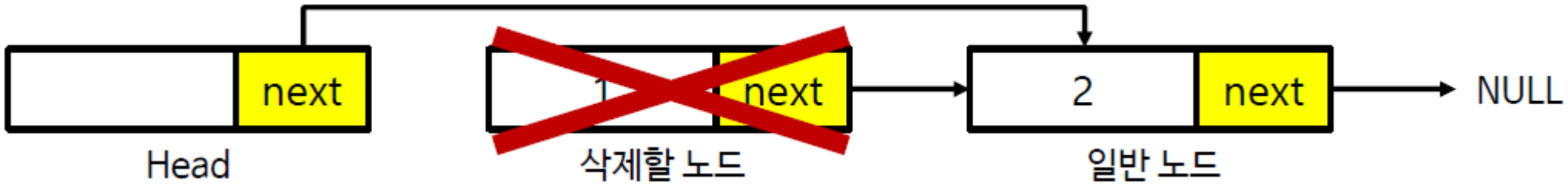


2.2 단순연결리스트

□ 삽입



□ 삭제



2.2 단순연결리스트 수행시간

□ 탐색 : 첫 노드부터 순차적으로 방문해야 하므로 O(N) 시간 소요

- 배열에 비해 검색 성능은 좋지 않음.

□ 삽입, 삭제 : 각각 상수 개의 레퍼런스를 갱신하므로 O(1) 시간이 소요

- 삽입, 삭제할 자료의 전후 노드의 참조 관계만 수정하면 되기 때문에 일정한 성능으로 동작. -> **효율적**
- 배열에 비해 구현은 복잡한 편.

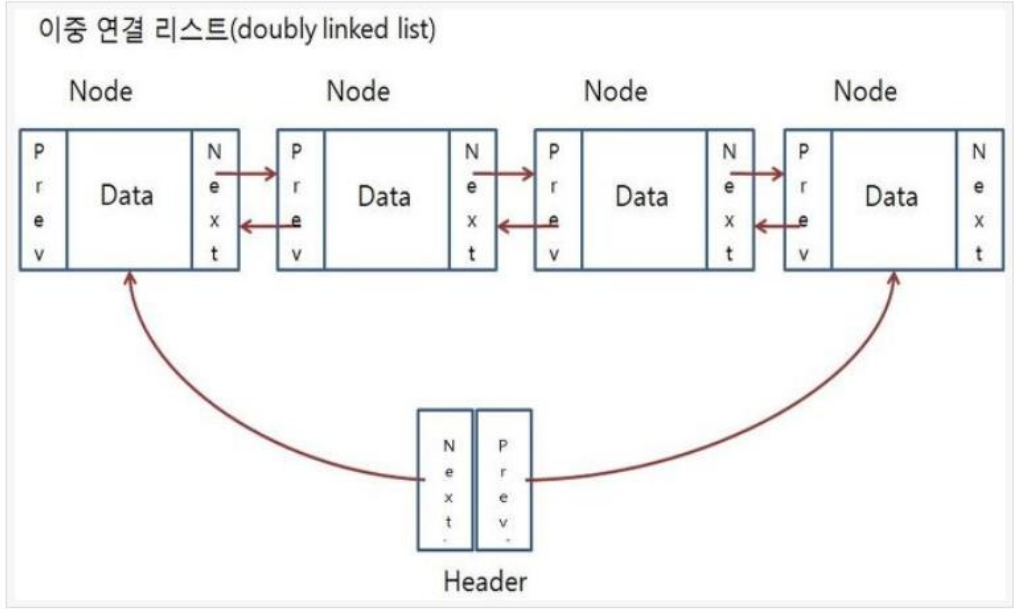
2.3 이중연결리스트

□ 이중연결리스트

- 각 노드가 두 개의 레퍼런스를 가지고 각각 이전 노드와 다음 노드를 가리키는 연결리스트.

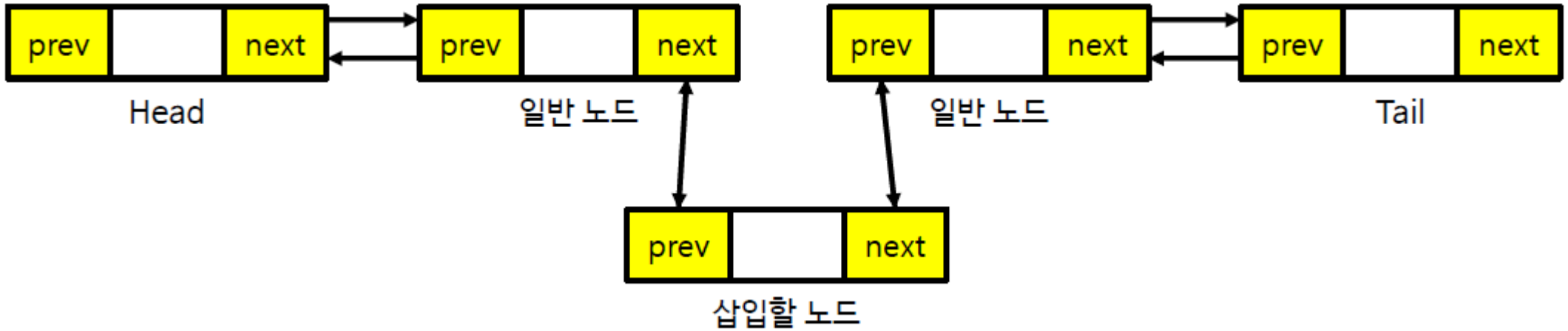
- 양방향으로 탐색이 가능하도록 만들어 탐색 속도를 향상시킬 수 있음.

Ex) 탐색하려는 데이터가 앞부분에 있을 경우 순방향, 뒷부분에 있을 경우 역방향으로 탐색. 약 2배 정도의 빠른 효율.

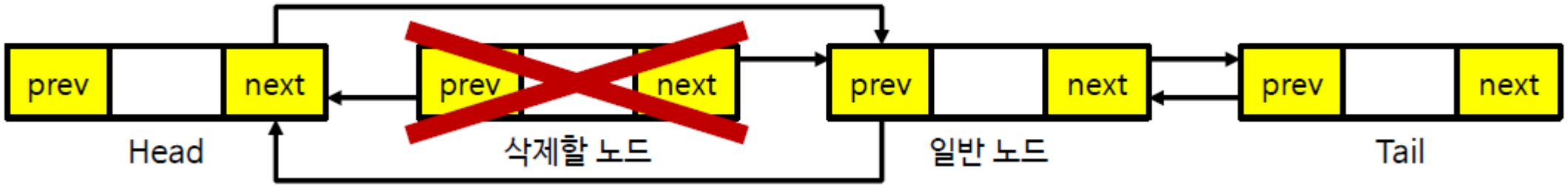


2.3 이중연결리스트

□ 삽입



□ 삭제



2.4 원형연결리스트

□ 원형 연결리스트

- 마지막 노드가 첫 노드와 원형으로 연결된 단순연결리스트.
- 계속 순회하면 이전 노드에 접근 할 수 있다.
- 마지막 노드와 첫 노드를 O(1) 시간에 접근할 수 있다.