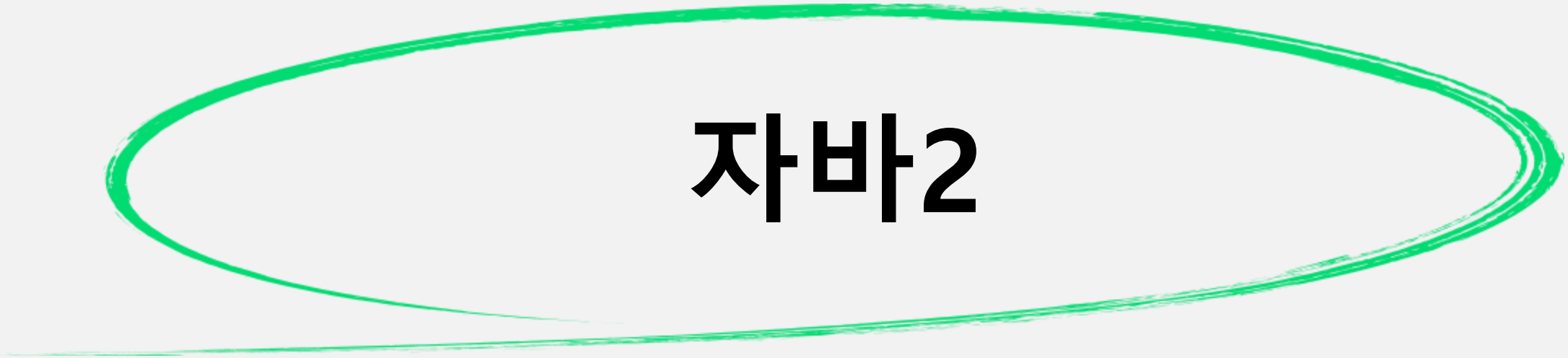


자바2

(알고리즘3 – Greedy/DP)

Chapter 05
알고리즘3 – Greedy/DP

Chapter05의 학습목표

- 탐욕 알고리즘과 동적 계획법의 특징에 대해 알아본다.
- 탐욕 알고리즘과 동적 계획법 관련 문제를 풀어보고 알고리즘을 이해한다.
- HashMap, HashSet 자료구조에 대해 학습한다

탐욕 알고리즘

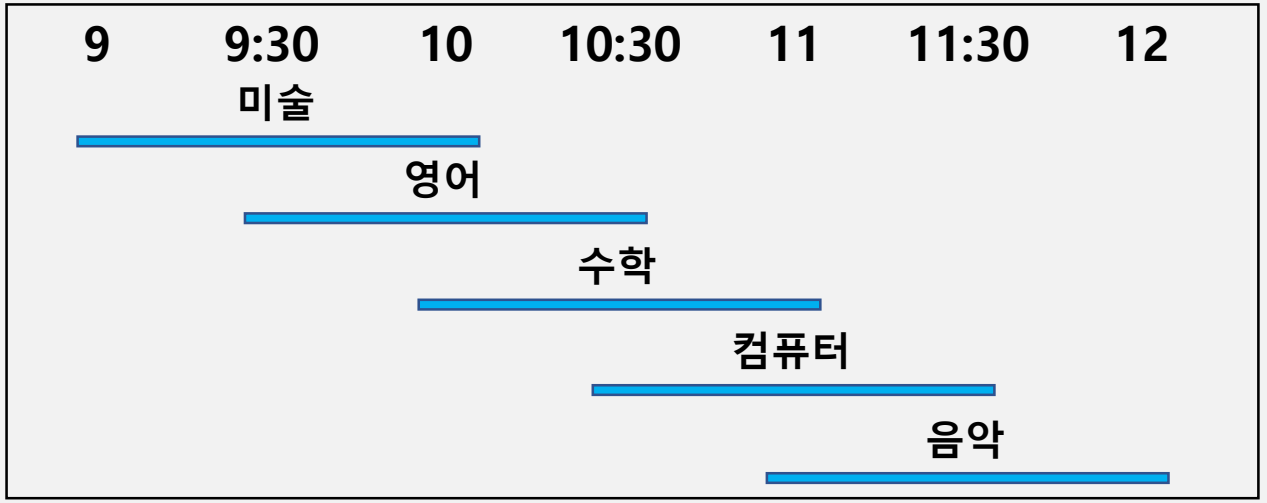
탐욕 알고리즘 (Greedy Algorithm)

- 최적해를 구하는 데에 사용되는 근사적인 방법으로, 여러 경우 중 하나를 결정해야 할 때마다 그 순간에 최적이라고 생각되는 것을 선택해 나가는 방식으로 진행하여 최종적인 해답에 도달하는 방식
- 순간마다 하는 선택은 그 순간에 대해 지역적으로는 최적이지만, 그 선택들을 계속 수집하여 최종적인 해답을 만들었다고 해서, 그것이 최적이라는 보장은 없음
- 따라서 탐욕알고리즘을 적용할 수 있는 문제들은 지역적으로 최적이면서 전역적으로 최적인 문제들임

수업 시간표

수업	시작	종료
미술	9 AM	10 AM
영어	9:30 AM	10:30 AM
수학	10 AM	11 AM
컴퓨터	10:30 AM	11:30 AM
음악	11 AM	12 PM

수업 시간표



탐욕 알고리즘

탐욕 알고리즘 (Greedy Algorithm)

가장 많은 과목을 신청할 수 있는 방법은?

1. 가장 빨리 끝나는 과목을 선택하여 신청
2. 첫 번째 과목이 끝난 후에 시작하는 과목을 골라 그 중에서 가장 빨리 끝나는 과목을 신청

1-2

수업	시작	종료
미술	9 AM	10 AM
영어	9:30 AM	10:30 AM
수학	10 AM	11 AM
컴퓨터	10:30 AM	11:30 AM
음악	11 AM	12 PM

3-4

수업	시작	종료
미술	9 AM	10 AM
영어	9:30 AM	10:30 AM
수학	10 AM	11 AM
컴퓨터	10:30 AM	11:30 AM
음악	11 AM	12 PM

5-6

수업	시작	종료
미술	9 AM	10 AM
영어	9:30 AM	10:30 AM
수학	10 AM	11 AM
컴퓨터	10:30 AM	11:30 AM
음악	11 AM	12 PM

탐욕 알고리즘

거스름돈 문제

손님에게 거스름돈을 거슬러 주려고 한다. 거스름돈이 770원인 경우, 동전의 개수가 최소가 되도록 동전을 선택하는 방법을 구하시오. 동전의 종류는 다음과 같다.
- 500원 / 100원 / 50원 / 10원

1. 가장 가치가 높은 동전을 선택한다.
2. 그 동전의 종류로 거슬러 줄 수 있는 만큼 수량을 선택한다.
3. 거스름돈을 전부 거슬러 줄 때까지 1로 돌아가서 반복한다.

반복1 : 500원을 선택 -> 500원 1개를 거슬러 줌 (남은 거스름돈 – 270)
반복2 : 100원을 선택 -> 100원 2개를 거슬러 줌 (남은 거스름돈 – 70)
반복3 : 50원을 선택 -> 50원 1개를 거슬러 줌 (남은 거스름돈 – 20)
반복3 : 10원을 선택 -> 10원 2개를 거슬러 줌 (남은 거스름돈 – 0)
종료

탐욕 알고리즘

거스름돈 문제

```
public static void main(String[] args) {
    //거슬러야 할 거스름돈
    int change = 770;
    //동전의 종류를 저장
    int[] coins = {10, 100, 50, 500};
    Arrays.sort(coins); //가격 순서대로 정렬
    //가장 가격이 큰 동전부터 거슬러주기 시작
    for (int i = coins.length - 1; i >= 0; i--) {
        //해당 동전으로 거슬러 줄 수 있다면
        while(change - coins[i] >= 0) {
            change -= coins[i];
            System.out.println(coins[i] +"원을 거슬러 줍니다.");
        }
    }
    System.out.println("남은 거스름돈 = " + change);
}
```

탐욕 알고리즘

거스름돈 문제

탐욕 알고리즘으로 거스름돈 문제를 풀었을 때의 문제점?

거스름돈이 340원이고, 동전의 종류가 다음과 같을 때
- 500원 / 100원 / 50원 / 70원/ 10원

최적 해 : 70원 4개(280) + 50원 1개(50) + 10원 1개(10)

탐욕 알고리즘의 해 : 100원 3개(300) + 10원 4개(40)

```
100을 거슬러 줍니다.
100을 거슬러 줍니다.
100을 거슬러 줍니다.
10을 거슬러 줍니다.
10을 거슬러 줍니다.
10을 거슬러 줍니다.
10을 거슬러 줍니다.
남은 거스름돈 = 0
```

탐욕 알고리즘

탐욕 알고리즘의 과정

1. 선택 과정 : 공집합에서 시작하여 집합에 추가할 다음 최적의 원소를 고른다.
2. 적절성 검사 : 새 집합이 해답으로 적절한지 검사한다.
3. 해답 점검 : 새 집합이 문제의 해답인지 판단한다.

탐욕 알고리즘의 장점과 단점

- 장점 : 상대적으로 설계하기 쉽다.

- 분할정복이나 동적계획법은 재귀적 관계를 발견해야 하는데 관계를 발견하기 어렵다.
- N이 굉장한 큰 문제에서는 분할정복이나 동적계획법보다 탐욕 알고리즘이 유리하다.

- 단점 : 최적화 문제에서 반드시 정확성을 증명해야 한다.

- 구한 해가 최적의 해인지 정확히 알 수 없으니 반드시 증명이 필요하다.
- 재귀적 관계에 의해 수학적으로 증명하는 것이 번거롭고 어렵다.

동적 계획법

동적 계획법(Dynamic Programming)

- 어떤 문제에 대한 최적해답을 얻을 때 문제를 부분적으로 분할하여, 각 부분에 대해 가장 적당한 해답을 차례로 구하는 것으로서 문제 전체에 대한 가장 적당한 해답을 얻는 알고리즘
- 작은 문제의 해답을 저장해 놓은 후, 큰 문제를 풀 때 작은 문제의 해답을 사용하여 큰 문제를 해결함
- 완전탐색과 비슷하지만 작은 문제의 해답을 저장한다는 점에서 완전탐색보다 속도가 빠르고, 한번 풀었던 문제를 다시 풀지 않아도 된다는 장점이 있음 (Memoization)

메모이제이션 (Memoization)
- 정답을 계산한 뒤, 계산된 값을 배열에 저장하는 방식

동적 계획법

피보나치 수열

- 처음 두 항을 1과 1로 한 후, 그 다음 항부터는 바로 앞의 두 개의 항을 더해 만드는 수열

  ex) 1, 1, 2, 3, 5, 8, 13, 21, 34, 55
- 피보나치 수열에 속한 수를 '피보나치 수'라고 함

피보나치 수열f(n)에서 n = 30의 값을 구하시오.

```java
public static void main(String[] args) {
    int n = 5;
    int[] fn = new int[n];
    fn[0] = 1;
    fn[1] = 1;
    for (int i = 2; i < fn.length; i++) {
        fn[i] = fn[i - 2] + fn[i - 1];
    }

    System.out.println(fn[n - 1]);
}
```

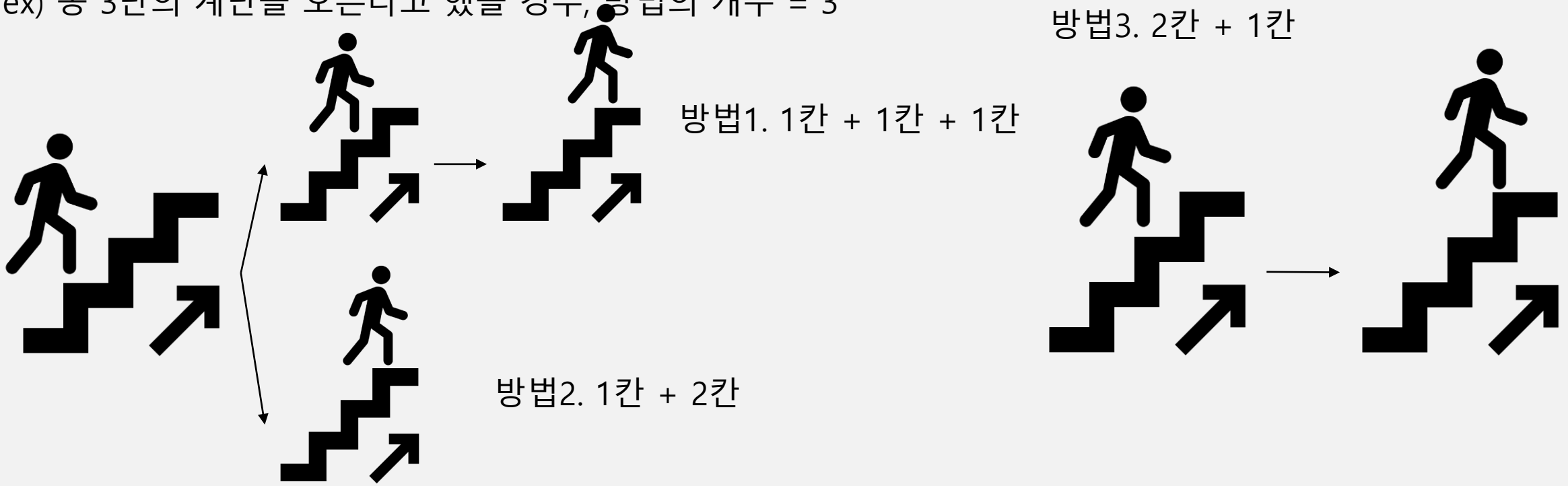
```java
static int pibonacci(int i1, int i2, int count, int n) {
    if(count > n) {
        return i1;
    }
    return pibonacci(i2, i1 + i2, count + 1, n);
}

public static void main(String[] args) {
    int n = 5;
    System.out.println(pibonacci(1, 1, 2, n));
}
```

동적 계획법

계단 오르기 문제

총 40단의 계단이 있다. 계단을 오를 때는 한 칸 씩, 또는 두 칸 씩 올라갈 수 있다.
40단까지 올라갈 경우 오를 수 있는 방법은 총 몇 개인가?

ex) 총 3단의 계단을 오른다고 했을 경우, 방법의 개수 = 3



동적 계획법

배낭 채우기 문제

배낭을 하나 가지고 가게에서 도둑질을 가기 위해 물건을 담으려고 한다. 그 배낭에는 총 35kg까지의 무게까지 담을 수 있다. 배낭에 넣을 물건의 가격의 합을 최대한 크게 하는 방법은 무엇인가. 가게에서 훔칠 수 있는 물건은 다음과 같다.

스테레오 : 300만원 / 30kg

노트북 : 250만원 / 20kg

기타 : 100만원 / 15kg

스테레오 하나를 넣을 경우

노트북과 기타를 넣을 경우

동적 계획법

배낭 채우기 문제 : 탐욕 알고리즘

스테레오 : 300만원 / 30kg
노트북 : 200만원 / 20kg
기타 : 150만원 / 15kg

탐욕 알고리즘을 사용
1) 가격이 가장 높은 스테레오를 선택 - *실패!*
2) *가장 가벼운 기타를 선택 – 성공!*
하지만 이어폰 : 100만원 /10kg 의 물건이 하나 추가된다면?
- 실패

1) 가격을 기준으로 잡아 스테레오 하나를 넣을 경우

2) 무게를 기준으로 잡아 노트북과 기타를 넣을 경우

3) 무게를 기준으로 잡았는데 물건이 추가되었을 경우

동적 계획법

배낭 채우기 문제 : 완전탐색 알고리즘

스테레오 : 300만원 / 30kg
노트북 : 200만원 / 20kg
기타 : 150만원 / 15kg

모든 물건의 조합을 시도해서 각 경우의 총 가치를 구해 가장 가치가 높은 경우를 선택 :
물건이 추가될 때마다 경우의 수가 2배가 됨 – 시간 복잡도는 $O(2^n)$

번호	조합	가격	무게	가방에 넣을 수 있는지 여부
1	스테레오	300만원	30kg	O
2	스테레오 + 노트북	500만원	50kg	X
3	스테레오 + 기타	450만원	45kg	X
4	노트북	200만원	20kg	O
5	**노트북 + 기타**	**350만원**	**35kg**	**O**
6	기타	150만원	15kg	O
7	스테레오 + 노트북 + 기타	650만원	65kg	X
8	물건을 넣지 않음	0만원	0kg	X

동적 계획법

배낭 채우기 문제 : 완전탐색 알고리즘

```
public class A_Class {
    static int[][] item = {{300, 30}, {200, 20}, {150, 15}};
    static ArrayList<Integer> cost_list = new ArrayList<>();

    static void brute_force(int curt_cost, int curt_weight, int start) {
        cost_list.add(curt_cost);
        for (int i = start; i < item.length; i++) {
            brute_force(item[i][0] + curt_cost, item[i][1] + curt_weight, i + 1);
        }
    }

    public static void main(String[] args) {
        brute_force(0, 0, 0);
        for (Integer cost : cost_list) {
            System.out.println(cost);
        }
    }
}
```

동적 계획법

배낭 채우기 문제 : 동적 계획법

- 더 작은 배낭 , 학위 배낭(sub-knapsack)에 대한 문제를 풀고 이를 이용해서 원래의 문제를 품

배낭을 격자(grid) 표로 나타냈을 경우 : Round1

현재 가방에 들어가는 최대 가격 : 150만원

선택할 물건/ 배낭 크기	15kg	20kg	25kg	30kg	35kg
기타 (150 / 15)	**150 / 15**	**150 / 15**	**150 / 15**	**150 / 15**	**150 / 15**
스테레오 (300 / 30)					
노트북 (200 / 20)					

동적 계획법

배낭 채우기 문제 : 동적 계획법

Round2

현재 가방에 들어가는 최대 가격 : 150만원

선택할 물건/ 배낭 크기	15kg	20kg	25kg	30kg	35kg
기타 (150 / 15)	**150 / 15**	150 / 15	150 / 15	150 / 15	150 / 15
스테레오 (300 / 30)	**150 / 15**				
노트북 (200 / 20)					

현재 15kg가방의 최대 가치

스테레오는 15kg 가방에 들어가지 않음

동적 계획법

배낭 채우기 문제 : 동적 계획법

Round2

현재 가방에 들어가는 최대 가격 : 300만원

선택할 물건/ 배낭 크기	15kg	20kg	25kg	30kg	35kg
기타 (150 / 15)	150 / 15	150 / 15	150 / 15	150 / 15	150 / 15
스테레오 (300 / 30)	150 / 15	150 / 15	150 / 15	**300 / 30**	**300 / 30**
노트북 (200 / 20)					

현재 10kg가방의 최대 가치

스테레오는 30kg 가방에 들어가고, 현재 최대 가치보다 가치가 큼! 갱신!

동적 계획법

배낭 채우기 문제 : 동적 계획법

Round3

현재 가방에 들어가는 최대 가격 : 350만원

선택할 물건/ 배낭 크기	15kg	20kg	25kg	30kg	35kg
기타 (150 / 15)	150 / 15	150 / 15	150 / 15	150 / 15	150 / 15
스테레오 (300 / 30)	150 / 15	150 / 15	150 / 15	300 / 30	300 / 30
노트북 (200 / 20)	150 / 15	**200 / 20**	**200 / 20**	300 / 30	**350 / 35**

동적 계획법

배낭 채우기 문제 : 동적 계획법

```
int[][] item = {{150, 15}, {300, 30}, {200, 20}};
int[] weight = {15, 20, 25, 30, 35};

int[][] cost_list = new int[item.length][weight.length];

int[][] knapsack_cost = new int[item.length][weight.length];
int[][] knapsack_weight = new int[item.length][weight.length];

for (int i = 0; i < cost_list.length; i++) {
    for (int j = 0; j < cost_list[0].length; j++) {
        if(i == 0 && item[i][1] <= weight[j]) {
            knapsack_cost[i][j] = item[i][0];
            knapsack_weight[i][j] = item[i][1];
        }
        else {
            knapsack_cost[i][j] = 0;
            knapsack_weight[i][j] = 0;
        }
    }
}
```

배낭에 첫 물건만 넣음

동적 계획법

배낭 채우기 문제 : 동적 계획법

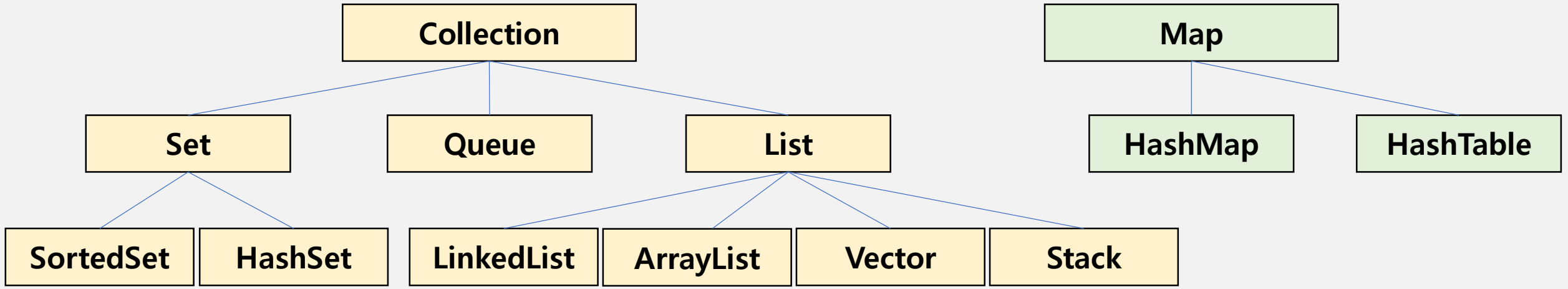
```
for (int i = 1; i < cost_list.length; i++) {
    for (int j = 0; j < cost_list[0].length; j++) {
        if(item[i][1] <= weight[j]) {
            knapsack_cost[i][j] = item[i][0];
            knapsack_weight[i][j] = item[i][1];

            for (int i2 = 0; i2 < i; i2++) {
                int new_cost = item[i][0] + knapsack_cost[i2][j];
                int new_weight = item[i][1] + knapsack_weight[i2][j];

                if(new_cost > knapsack_cost[i][j] && new_weight <= weight[j]) {
                    knapsack_cost[i][j] = item[i][0] + knapsack_cost[i2][j];
                    knapsack_weight[i][j] = item[i][1] + knapsack_weight[i2][j];
                }
            }
        }
    }
}
```

배낭에 물건을 넣으면서 기존 배열을 재탐색, 최적의 답을 탐색함

Collection

Collection Framework

- 여러 가지 자료 구조를 자바에서 미리 구현하여 제공하는 클래스
- 객체만 저장이 가능
- 크게 Collection계열과 Map계열로 나뉨



Collection

Collection 계열 클래스의 기능 (메소드)

메소드	설명
boolean add(E e)	객체(데이터)를 collection에 추가
void clear()	
boolean contains(Object o)	현재 collection에 인자로 전달된 데이터가 있는지 판별
boolean isEmpty()	현재 collection에 데이터가 있는지 검사
Iterator<E> iterator()	현재 collection의 객체를 iterator로 반환
boolean remove(Object o)	현재 collection에 인자로 전달된 객체를 제거
int size()	현재 collection의 데이터 개수를 반환
Object[] toArray()	현재 collection이 가지고 잇는 객체를 배열로 만들어 반환

Collection

Iterator / Enumeration

- Collection 클래스에 저장된 데이터를 일정한 방법으로 접근할 수 있도록 함
- 각 컬렉션 클래스에 저장할 때처럼 가져올 때도 일관성 있게 가져오자는 개념에서 나온 클래스
- Iterator는 모든 클래스에서 사용이 가능

```
Iterator iterator = <V>.iterator();
```

클래스명	메소드명	반환 타입	설명
Iterator	hasNext()	boolean	iteration이 element를 가지고 있는지 판별
	next()	E	iteration의 다음 element를 반환
	remove()	void	iteration이 마지막으로 반환한 element다음의 element를 제거
Enumeration	hasMoreElements()	boolean	enumeration이 추가의 element를 가지고 있는지 판별
	nextElement()	E	enumeration이 다른 element를 가지고 있으면 element를 반환

Collection

Set - HashSet

- Set은 집합이라는 뜻으로, 요소들을 집합적으로 모아놓은 자료구조를 의미
- 중복된 데이터를 가지지 않으며 저장 순서를 유지하지 않는다는 특징이 있음

```
HashSet<E> 변수명 = new HashSet<>(); //순서가 없음
HashSet<E> 변수명 = new LinkedHashSet<>(); //순서를 유지
```

메소드명	반환 타입	설명
add(값)	boolean	HashSet에 값을 추가. 해당 값이 이미 존재한다면 추가하지 않고 false를 반환
remove(값)	boolean	해당 값이 HashSet에 존재한다면 삭제 후 true를 반환, 없다면 false를 반환
clear()	-	HashSet에 있는 모든 값을 제거
size()	int	HashSet의 데이터 크기(개수)를 반환
next()	object	HashSet의 다음 데이터를 가져옴
hasNext()	Boolean	HashSet안에 데이터가 있다면 true를, 없다면 false를 반환
iterator()	iterator	Iterator 반복자를 사용하여 데이터를 순회
contains(값)	Boolean	HashSet안에 해당 값이 있다면 true, 없다면 false를 반환

Collection

HashSet

```
HashSet<String> hashSet = new HashSet<>();
hashSet.add("demon");
hashSet.add("banana");
hashSet.add("tomato");
hashSet.add("apple");
hashSet.add("cargo");

hashSet.add("apple");
hashSet.add("banana");

Iterator iterator = hashSet.iterator();
while(iterator.hasNext()) {
    System.out.println(iterator.next());
}
```

```
banana
apple
demon
tomato
cargo
```

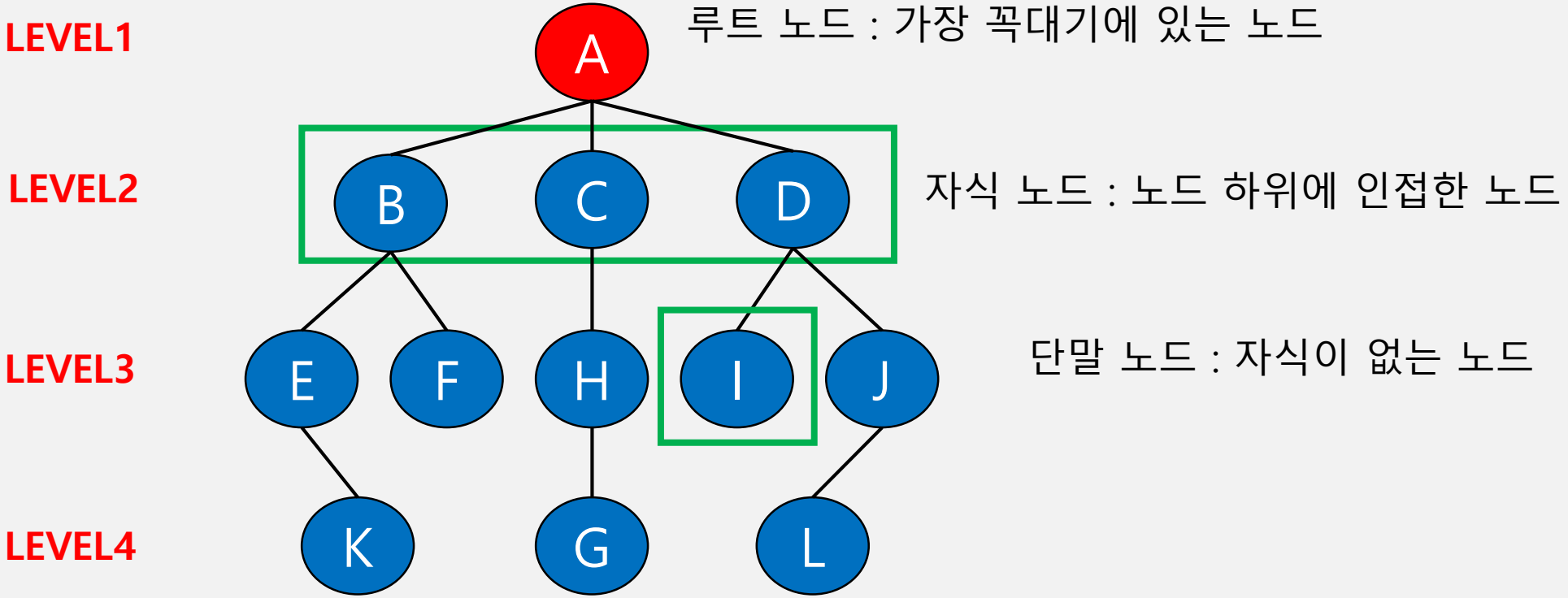
```
HashSet<String> hashSet = new LinkedHashSet<>();

hashSet.add("demon");
hashSet.add("banana");
hashSet.add("tomato");
hashSet.add("apple");
hashSet.add("cargo");

hashSet.add("apple");
hashSet.add("banana");

Iterator iterator = hashSet.iterator();
while(iterator.hasNext()) {
    System.out.println(iterator.next());
}
```

```
demon
banana
tomato
apple
cargo
```

Collection

트리 (Tree)

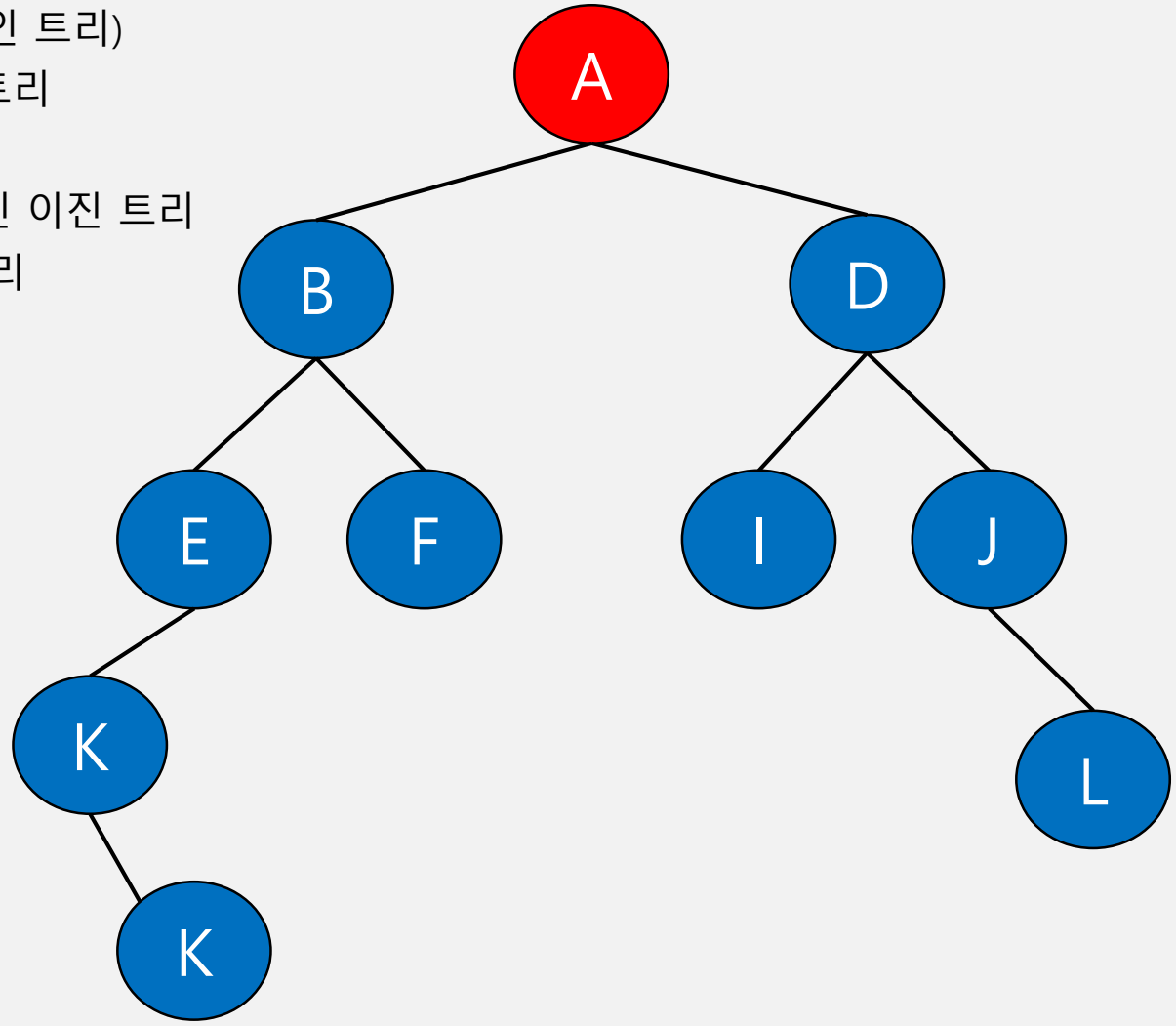
- 루트 노드를 시작으로 줄기를 뻗어 나가듯 원소를 추가해 나가는 비선형 구조
- 방향성이 있고 노드와 노드 간에 간선이 존재하여 간선으로 연결됨
- 차수 (Degree) : 노드의 하위 간선의 개수 (자식의 개수 = 간선의 개수)
- 깊이 (depth) : 루트에서 어떤 루트까지의 경로 길이



Collection

이진 탐색 트리 (BST : Binary Search Tree)

- 트리의 최대 차수가 2인 트리 (모든 노드의 자식이 최대 2개인 트리)
- 부모 노드보다 작으면 왼쪽, 크면 오른쪽 자식 노드가 되는 트리
- 중복된 데이터를 허용하지 않음
- 완전 이진 트리 : 마지막 노드를 제외하고 모든 노드가 채워진 이진 트리
- 포화 이진 트리 : 모든 단말 노드의 깊이가 같은 완전이진 트리
- 균형 이진 트리 : 좌우의 높이의 차가 같거나 최대 1인 트리



Collection

Map – TreeSet

- 중복 불가, 순서가 없고 데이터 정렬 기능을 제공함
- 이진 탐색 트리의 구조로 이루어져 있기 때문에 데이터 검색에 유리하다는 특징이 있음

메소드명	반환 타입	설명
ceiling(E e)	E	인자로 전달되는 element보다 작은 element를 반환
descendingIterator()	Iterator<E>	element들을 set 타입으로 정렬하는데 사용되는 Comparator 객체를 반환
first()	E	set에 있는 가장 첫 번째 element를 반환
floor(E e)	E	인자로 전달되는 element보다 큰 element 중 가장 큰 element를 반환
higher(E e)	E	인자로 전달되는 element보다 큰 element 중 가장 작은 element를 반환
last()	E	set에서 가장 마지막 element를 반환
tailSet(E fElement)	SortedSet<E>	set에서 fElement보다 크거나 같은 element들을 정렬한 후 set 타입으로 반환
subSet(E fElement, E tElement)	SortedSet<E>	fElement와 tElement 사이의 element들을 정렬한 후 set 타입으로 반환
headSet(E toElement)	SortedSet<E>	set에서 toElement보다 작은 element들을 정렬한 후 set 타입으로 반환

Collection

Map – TreeSet

```
public static void main(String[] args) {
    TreeSet ts = new TreeSet();
    ts.add("홍길동");
    ts.add("차범근");
    ts.add("유재석");
    ts.add("박명수");
    ts.add("김유신");
    ts.add("홍길동");

    Iterator ite = ts.iterator();
```

```
    System.out.println("오름차순 정렬");
    while(ite.hasNext()) {
        System.out.println(ite.next());
    }

    System.out.println("\n내림차순 정렬");
    Iterator ite2 = ts.descendingIterator();
    while(ite2.hasNext()) {
        System.out.println(ite2.next());
    }

    System.out.println(ts);
}
```

Collection

Map - HashMap

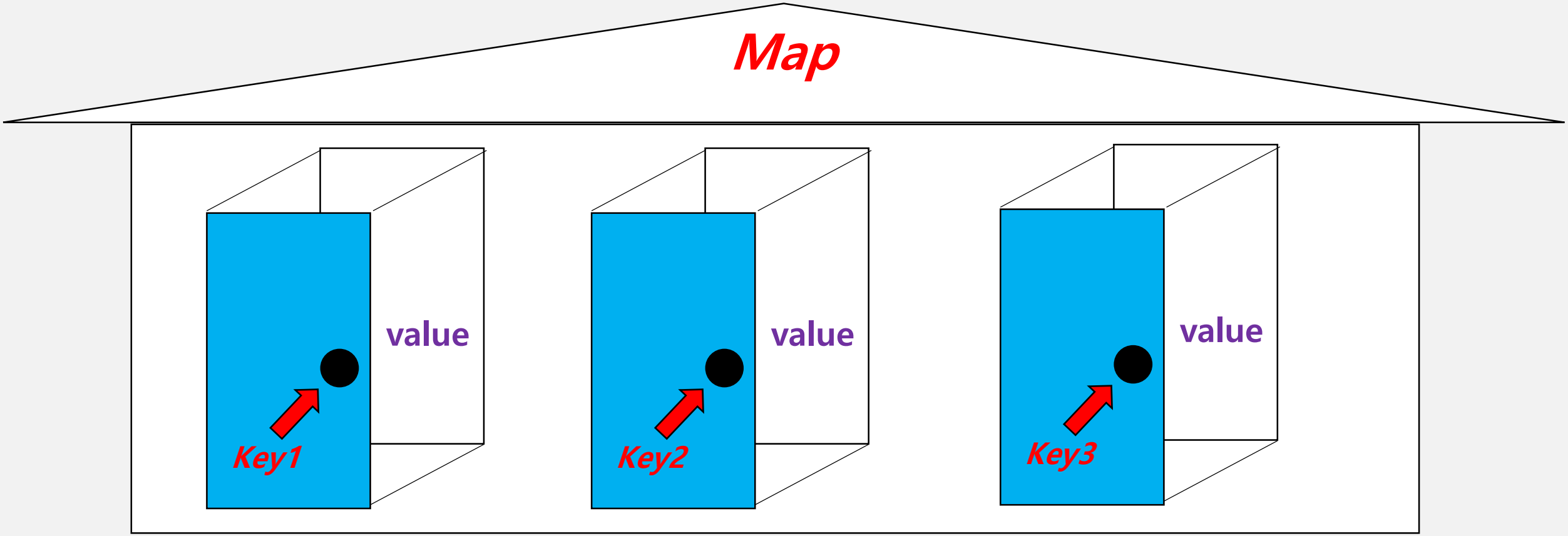
- Map 인터페이스는 키와 값을 쌍으로 저장하는 구조로, key값은 중복이 허용되지 않지만 value 값은 중복이 허용됨
- HashMap은 내부적으로 해싱(Hashing) 검색을 사용하기 때문에 대용량 데이터관리에 성능이 좋음

```
HashMap<Key, Value> hashMap = new HashMap<>();
```

메소드명	반환 타입	설명
containsKey(값)	boolean	HashMap 안에 해당 key가 있는지 여부를 반환
containsValue(값)	boolean	HashMap 안에 해당 value가 있는지 여부를 반환
get(key)	object	key의 쌍인 value값을 반환
values	Collection	HashMap에 존재하는 value를 Collection 타입으로 반환
Set<key> KeySet()	Set	키들을 Set형태로 반환
put(key, value)		key와 value 한 쌍을 HashMap에 추가
remove(key)		HashMap에서 해당 key를 가진 쌍을 제거
size()	int	HashMap 전체 요소의 개수를 반환

Collection

Map



Collection

HashMap

```
Scanner input = new Scanner(System.in);
HashMap<String, Integer> hashMap = new HashMap<>();

hashMap.put("종이", 100);
hashMap.put("연필", 1000);
hashMap.put("가위", 3000);

System.out.println("가격을 알고싶은 물건을 입력하세요 >>> ");
String object = input.next();
if(hashMap.containsKey(object)) {
    int price = hashMap.get(object);
    System.out.println(object + "는 " + price + "원 입니다.");
}
else {
    System.out.println("해당 물건은 존재하지 않습니다.");
}
```

Collection

Map – TreeMap

- 중복 불가, 순서가 없고 데이터 정렬 기능을 제공함 – 저장 시 내부적으로 key를 오름차순 정렬하여 저장
- 이진 탐색 트리의 구조로 이루어져 있기 때문에 데이터 검색에 유리하다는 특징이 있음

메소드명	반환 타입	설명
ceilingKey(K key)	K	인자로 전달되는 element보다 크거나 같은 key 중 가장 작은 key를 반환
firstKey()	K	key에 있는 가장 첫 번째(가장 작은) key를 반환
firtstEntry()	Map.Entry<K, V>	key에 있는 가장 첫 번째(가장 작은) key를 반환
descendingKeySet()	NavigableSet<K>	map에 존재하는 key의 역순으로 된 NavigableSet을 반환
headMap(K toKey)	SortedMap<K,V>	map에서 toKey보다 작은 key들을 정렬한 후 반환
lastKey()	K	key에서 가장 마지막 element를 반환
lowerKey(K key)	K	map에서 인자로 전달된 key보다 작은 key 중 가장 큰 key를 반환
subMap(K fK, K tKey)	SortedMap<K,V>	map에서 fKey와 tKey 사이의 키 값을 가지는 key들을 반환
tailMap(K fKey)	SortedMap<K,V>	map에서 fKey보다 크거나 같은 key들을 반환

HashMap

https://programmers.co.kr/learn/courses/30/lessons/42576

완주하지 못한 선수

수많은 마라톤 선수들이 마라톤에 참여하였습니다. 단 한 명의 선수를 제외하고는 모든 선수가 마라톤을 완주하였습니다. 마라톤에 참여한 선수들의 이름이 담긴 배열 participant와 완주한 선수들의 이름이 담긴 배열 completion이 주어질 때, 완주하지 못한 선수의 이름을 출력하세요.

- 마라톤 경기에 참여한 선수의 수는 1명 이상 100,000명 이하입니다.
- completion의 길이는 participant의 길이보다 1 작습니다.
- 참가자의 이름은 1개 이상 20개 이하의 알파벳 소문자로 이루어져 있습니다.
- 참가자 중에는 동명이인이 있을 수 있습니다.

participant	completion	출력
["leo", "kiki", "eden"]	["eden", "kiki"]	"leo"
["marina", "josipa", "nikola", "vinko", "filipa"]	["josipa", "filipa", "marina", "nikola"]	"vinko"
["mislav", "stanko", "mislav", "ana"]	["stanko", "ana", "mislav"]	"mislav"

알고리즘/자료구조

이 외 알고리즘

자료구조 / 알고리즘 종류	관련 알고리즘 / 문제
완전 탐색 (Brute-force)	에라토스테네스의 체(소수 판별)
분할 정복(divide and conquer)	이진 탐색
해쉬(Hash), 보안 알고리즘	공통키 암호, 공개키 암호, 하이브리드 암호, 전자 서명
Heap, Queue	Heap Sort, 우선순위 큐, 이중 우선순위 queue
백 트래킹 (backtracking), 그래프(Graph)	플로이드 워셜 알고리즘, 프림 알고리즘, 여왕 문제(N-QEENS), A*(A Star 알고리즘)
DFS, BFS, 재귀	길 찾기, 하노이의 탑
트리	이진 트리, TreeMap, TreeHash

DP / Greedy / Hash

추천 문제

DP (Dynamic Programming)

1. https://programmers.co.kr/learn/courses/30/lessons/43105 (정수 삼각형)
2. https://programmers.co.kr/learn/courses/30/lessons/42895 (N으로 표현)
3. https://programmers.co.kr/learn/courses/30/lessons/42898 (등굣길)

Greedy Algorithm

1. https://programmers.co.kr/learn/courses/30/lessons/42862 (체육복)
2. https://programmers.co.kr/learn/courses/30/lessons/42885 (구명 보트)
3. https://programmers.co.kr/learn/courses/30/lessons/42883 (큰 수 만들기)

Hash

1. https://programmers.co.kr/learn/courses/30/lessons/42578 (위장)

수고하셨습니다.