Chapter 05.
재귀 #3

Clip 01 | [1182] 부분 수열의 합
부분 수열의 조합의 수

Clip 02 | [2758] 로또
귀납적 접근과 메모이제이션

Clip 03 | [1208] 부분 수열의 합 2
분할과 최소 경우의 수

Clip 04 | [1759] 암호 만들기
길이가 고정된 부분 수열

Clip 05 | [10971] 외판원 순회2
재귀를 이용한 완전탐색

Clip 06 | [14888] 연산자 끼워넣기
음수의 나눗셈과 경우의 수 조합

Clip 07 | [16987] 계란으로 계란치기
객체의 관리와 구현

Ch05. 재귀 #3

1. [1182] 부분 수열의 합

BOJ1182: 부분 수열의 합

문제

N개의 정수로 이루어진 수열이 있을 때, 크기가 양수인 부분수열 중에서 그 수열의 원소를 다 더한 값이 S가 되는 경우의 수를 구하는 프로그램을 작성하시오.

입력

첫째 줄에 정수의 개수를 나타내는 N과 정수 S가 주어진다. (1 ≤ N ≤ 20, |S| ≤ 1,000,000) 둘째 줄에 N개의 정수가 빈 칸을 사이에 두고 주어진다. 주어지는 정수의 절댓값은 100,000을 넘지 않는다.

부분 수열 원소의 합이 특정 값 S가 되는 경우의 수

BOJ1182: 부분 수열의 합

부분 수열?

부분 문자열(Substring) : 연속성을 가진다
부분 수열 (Subsequence) : 연속성을 가지지 않아도 된다

ex) ABCDEF의 BCD : Substring(O) Subsequence(O)
ABCDEF의 ACF : Substring(X) Subsequence(O)

BOJ1182: 부분 수열의 합

부분 수열의 모든 경우의 수

-7	-3	-2	5	8

- { i } 번째 수를 고르는 경우
- { i } 번째 수를 고르지 않는 경우

2^5

BOJ1182: 부분 수열의 합

Base Case

- 수열의 모든 수에 대해 뽑을지 말지 모두 판단한 경우

Recursive Case

- {index}번째 숫자까지의 결정에 대해 합을 알고 있을 때
  - {index+1} 번째 수를 포함하는 경우
  - {index+1} 번째 수를 포함하지 않는 경우

BOJ1182: 부분 수열의 합

Base Case

- 수열의 모든 수에 대해 뽑을지 말지 모두 판단한 경우

Recursive Case

- {index}번째 숫자까지의 결정에 대해 합을 알고 있을 때
  - {index+1} 번째 수를 포함하는 경우
  - {index+1} 번째 수를 포함하지 않는 경우

```
public static void solve(int index, int sum) {
    if(index == numbers.length) return;
    if(sum + numbers[index] == s) answer++;

    solve(index + 1, sum + numbers[index]);
    solve(index + 1 , sum);
}
```

Base Case

Recursive Case

BOJ1182: 부분 수열의 합

정답 계산

```
public static void solve(int index, int sum) {
    if(index == numbers.length) return;
    if(sum + numbers[index] == s) answer++;

    solve(index + 1, sum + numbers[index]);
    solve(index + 1 , sum);
}
```

index 번째 수를 고르는 경우?
→ sum + numbers[index]

index 번째 수를 고르지 않는다면?
현재 정답 상태를 이전 재귀에서 answer++ 하였기 때문에 패스

Ch05. 재귀 #3

2. [2758] 로또

BOJ2758: 로또

문제 요약

- [1, m] 범위의 정수에서 n개를 뽑는 순열
- 단, {i} 번째 수는 {i-1} 번째 수보다 두배 이상으로 커야 한다
- (1 <= n <= 10), (1<= m <= 2,000) (n <= m)

BOJ2758: 로또

큰 문제를 작은 문제로 쪼개면서 접근해보자

현재 {i} 번째 수에 대해서, 마지막 값이 {last} 인 경우

→ {i-1} 번째 수에 대해서 {last}/2 값을 고른 경우

→ {i} 번째 수를 {last}-1 로 골라도 되는데 증가시킨 경우

BOJ2758: 로또

큰 문제를 작은 문제로 쪼개면서 접근해보자

현재 {i} 번째 수에 대해서, 마지막 값이 {last} 인 경우

→ {i-1} 번째 수에 대해서 {last}/2 값을 고른 경우

→ {i} 번째 수를 {last}-1 로 골라도 되는데 증가시킨 경우


```
public static long solve(int i, int last) {
    if(last <= 0) return 0;
    return solve(i - 1, last / 2) + solve(i, last - 1);
}
```

solve는 수학적 의미의 함수인가?
→ 함수 내부에 상태를 저장하지 않는다
→ 반복 호출해도 항상 동일한 결과가 나온다

BOJ2758: 로또

solve는 수학적 의미의 함수이다
→ 한번 구한 결과는 저장해두고, 반복하지 않도록 처리한다

```
public static long[][] mem; // main 함수에서 -1 로 초기화
public static long solve(int i, int last) {
    if(last <= 0) return 0;
    if(mem[i][last] == -1) { // 한번도 계산한 적이 없다면 재귀 수행
        mem[i][last] = solve(i - 1, last / 2) + solve(i, last - 1);
    }
    return mem[i][last];
}
```

BOJ2758: 로또

{i} == 1인 경우는?

(현재 {i} 번째 수에 대해서, 마지막 값이 {last} 인 경우)

가능한 경우의 수가 {last} 만큼 나온다

마지막 값이 1보다 작은 수열 : {1}
마지막 값이 2보다 작은 수열 : {1}, {2}
마지막 값이 5보다 작은 수열 : {1}, {2}, {3}, {4}, {5}

따라서 {i} == 1 인 경우에는 {last} 값을 반환한다

```
if(i == 1) return last;
```

BOJ2758: 로또

```
public static long solve(int i, int last) {
    if(last <= 0) return 0;
    if(i == 1) return last;
    if(mem[i][last] == -1) {
        mem[i][last] = solve(i - 1, last / 2) + solve(i, last - 1);
    }
    return mem[i][last];
}
```

Base Case

Recursive Case

Ch05. 재귀 #3

3. [1208] 부분 수열의 합 2

BOJ1208: 부분 수열의 합 2

부분수열의 합 2 성공

☆

시간 제한	메모리 제한	제출	정답	맞힌 사람	정답 비율
1 초	256 MB	24524	6257	4186	24.452%

문제

N개의 정수로 이루어진 수열이 있을 때, 크기가 양수인 부분수열 중에서 그 수열의 원소를 다 더한 값이 S가 되는 경우의 수를 구하는 프로그램을 작성하시오.

입력

첫째 줄에 정수의 개수를 나타내는 N과 정수 S가 주어진다. ($1 \le N \le 40$, $|S| \le 1,000,000$) 둘째 줄에 N개의 정수가 빈 칸을 사이에 두고 주어진다. 주어지는 정수의 절댓값은 100,000을 넘지 않는다.

부분 수열 원소의 합이 특정 값 S가 되는 경우의 수
N의 범위가 40까지 확장됐다

BOJ1208: 부분 수열의 합 2

부분 수열의 모든 경우의 수

-7	-3	-2	5	8

- {i} 번째 수를 고르는 경우
- {i} 번째 수를 고르지 않는 경우

2^5

N == 40인 경우에는? 2^{40} 만큼의 조합이 발생한다
연산 횟수: 1,099,511,627,776 (약 3시간 소요, 초당 1억 회)

BOJ1208: 부분 수열의 합 2

수열의 범위를 두개의 구간으로 나눠서 생각해보자

임의의 값 {k}를 기준으로

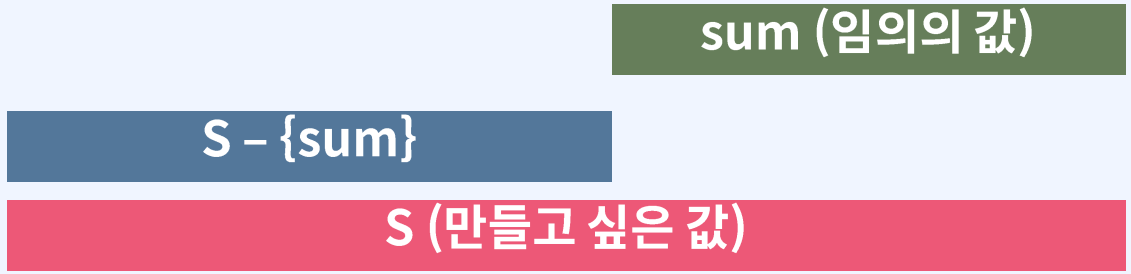
- LEFT: [0, k]
- RIGHT: [k, n]

두개의 구간으로 수열을 나눌 수 있다

BOJ1208: 부분 수열의 합 2

- LEFT: [0, k]
- RIGHT: [k, n]

만약 RIGHT 구간에서 임의의 수열의 합 {sum} 을 구했다고 가정해보자
이때 LEFT 구간에 S - {sum} 에 해당하는 경우의 수가 존재한다면?



S-{sum}을 만드는 각각의 경우에 {sum}을 더하면 S가 된다
→ S-{sum} 경우의 수만큼 S를 만들 수 있다

BOJ1208: 부분 수열의 합 2

S-{sum}을 만드는 각각의 경우에 {sum}을 더하면 S가 된다
→ S-{sum} 경우의 수만큼 S를 만들 수 있다

1. LEFT 구간에서 만들어지는 수열의 합을 Map에 저장해 둔다

2. RIGHT 구간에서 만들어지는 수열의 합 sum을 구한다

3. (1)의 Map을 통해 S-{rsum} 을 만드는 경우의 수를, S의 경우의 수에 더한다

4. (2)~(3)을 재귀를 통해 반복한다 (RIGHT구간의 모든 수열 만큼)

BOJ1208: 부분 수열의 합 2

임의의 값 {k}를 기준으로
- LEFT: [0, k]
- RIGHT: [k, n]

두개의 구간으로 수열을 나눌 수 있다
k는 어떤 값을 사용 하는게 좋을까?

(k == 1): LEFT: 2^1, RIGHT: $2^{n-1} \mid 2^1 + 2^{n-1}$

(k == n/2): LEFT: $2^{n/2}$, RIGHT: $2^{n/2} \mid 2^{n/2} + 2^{n/2}$

(k == n – 1): LEFT: 2^{n-1}, RIGHT: $2^1 \mid 2^{n-1} + 2^1$

k == n / 2 를 사용하면 가장 적은 경우의 수가 나온다. 왜 일까?

BOJ1208: 부분 수열의 합2

LEFT: 2^k, RIGHT: $2^{n-k} \mid 2^k + 2^{n-k}$

k에 대해 미분하면?

$$\frac{d}{dk} 2^k = \ln 2 \cdot 2^k$$

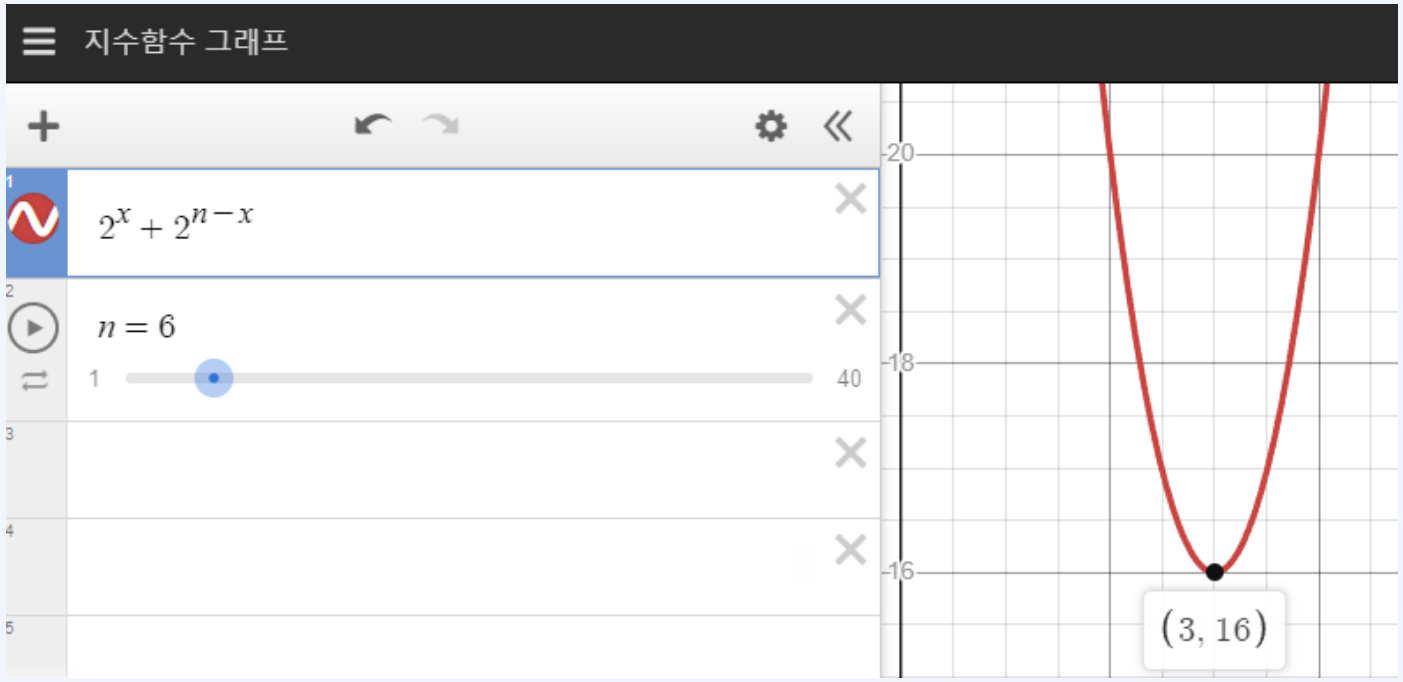
$$\frac{d}{dk} 2^{n-k} = -\ln 2 \cdot 2^{n-k}$$

따라서 $\{\ln 2 \cdot 2^k - \ln 2 \cdot 2^{n-k}\} = 0$ 을 만드는 k가 변곡점이 된다

$2^k = 2^{n-k}$

n과 k는 양의 정수범위로 k == n-k 를 만족시키려면 k == n/2가 된다

BOJ1208: 부분 수열의 합2



혹은 n과 k에 숫자를 대입하면서 그래프를 그려봐도 된다

BOJ1208: 부분 수열의 합

Base Case

- 수열의 모든 수에 대해 뽑을지 말지 모두 판단한 경우

Recursive Case

- {index - 1}번째 숫자까지의 결정에 대해 합을 알고 있을 때
  - {index} 번째 수를 포함하는 경우
  - {index} 번째 수를 포함하지 않는 경우

```
public static void solve(int index, int end, int sum) {
    if(index == end) { }

    else {
        solve(index + 1, end, sum);
        solve(index + 1, end, sum + numbers[index]);
    }
}
```

Base Case

Recursive Case

BOJ1208: 부분 수열의 합2

```
status = LEFT;
solve(0, n / 2, 0);

status = RIGHT;
solve(n / 2, n, 0);
```

```
public static Map<Integer, Integer> cnt = new HashMap<>();
public static void solve(int index, int end, int sum) {
    if(index == end) {
        if (status == LEFT) {
            int prev = cnt.getOrDefault(sum, 0);
            cnt.put(sum, prev + 1);
        } else if (status == RIGHT) {
            answer += cnt.getOrDefault(s - sum, 0);
        }
    }
    else {
        solve(index + 1, end, sum);
        solve(index + 1, end, sum + numbers[index]);
    }
}
```

Ch05. 재귀 #3

4. [1759] 암호 만들기

BOJ1759: 암호 만들기

문제 요약

- C개의 문자를 이용해서 L길이의 암호를 만들기
- 최소 한 개의 모음, 최소 두개의 자음
- 문자는 알파벳 소문자, 중복 없음
- 3 <= L <= C <= 15
- 암호는 알파벳이 증가하는 순서로 배열

BOJ1759: 암호 만들기

문제 분석

- 모음을 고려하지 않으면 C_L^C개의 경우의 수가 나온다

- C와 L은 모두 15 이하의 수이므로
  최악의 경우 $C_7^{15} = C_8^{15}$ = 6,435

- 따라서 모든 경우의 수를 다 구하고,
  모음 조건을 만족하는지 판단해도 충분하다

BOJ1759: 암호 만들기

Base Case:

- 암호가 원하는 길이를 만족했을 때

Recursive Case:

- 암호의 {length} 번째 문자를 input[index]로 고르는 경우
- 암호의 {length} 번째 문자를 input[index]로 고르지 않는 경우

BOJ1759: 암호 만들기

Base Case:
- 암호가 원하는 길이를 만족했을 때

Recursive Case:
- 암호의 {length} 번째 문자를 input[index]로 고르는 경우
- 암호의 {length} 번째 문자를 input[index]로 고르지 않는 경우


```
public static boolean isVowel(char c) {
    return c == 'a' || c == 'e' || c == 'i' || c == 'o' || c == 'u';
}
```

```
public static void generate(int length, int index, int vowelCnt) {
    if(length == l) { return; }
    if(index < c) {
        password[length] = input[index];
        int v = isVowel(input[index]) ? 1 : 0;
        generate(length + 1, index + 1, vowelCnt + v);
        password[length] = 0;
        generate(length, index + 1, vowelCnt);
    }
}
```

Base Case

Recursive Case

Recursive Case

BOJ1759: 암호 만들기

```
Arrays.sort(input);
```

```
public static void generate(int length, int index, int vowelCnt) {
    if(length == l) {
        if (vowelCnt >= 1 && l - vowelCnt >= 2) {
            System.out.println(password);
        }
        return;
    }
}
```

처음 받은 입력을 정렬했다면, base case에서 바로 출력을 진행해도 된다
(모음 조건 검사 포함)

Ch05. 재귀 #3

5. [10971] 외판원 순회 2

BOJ10971: 외판원 순회2

문제 요약

- N개의 도시 (2 <= N <= 10)
- 모든 도시를 방문하고 출발 정점으로 복귀
- 간선마다 cost가 존재
- 순회 가능한 경로 중 최소 cost를 사용하는 경로 구하기

BOJ10971: 외판원 순회2

문제 분석

NP-hard로 유명한 문제
동적계획법으로 최적화가 가능하지만, 아직은 다루지 않았다

N이 10 이하이므로 완전탐색으로 접근
→ 10! 의 경우의 수: 3,628,800

충분히 1초 이내에 연산할 수 있는 수준이다

BOJ10971: 외판원 순회2

인접행렬?

{i} – {j} 의 연결관계를 행렬로 표현한 것

	[1]	[2]	[3]	[4]
[1]	0	10	15	20
[2]	5	0	9	10
[3]	6	13	0	12
[4]	8	8	9	0

BOJ10971: 외판원 순회2

	[1]	[2]	[3]	[4]
[1]	0	10	15	20
[2]	5	0	9	10
[3]	6	13	0	12
[4]	8	8	9	0

src:{1} → dist:{2} 간선의 cost는 ?

BOJ10971: 외판원 순회2

	[1]	[2]	[3]	[4]
[1]	0	10	15	20
[2]	5	0	9	10
[3]	6	13	0	12
[4]	8	8	9	0

src:{i} → dist:{i} 간선의 cost는 ?

BOJ10971: 외판원 순회2

인접행렬의 특징

- 그래프의 방향 표현 가능
  - [i][j] / [j][i]
- n개의 정점 → n^2 의 공간 복잡도
- 간선 조회 / 저장 시간 복잡도: O(1)
  - 특정 정점의 모든 간선 조회: O(n)

BOJ10971: 외판원 순회2

Base Case

- 모든 도시(정점)을 모두 방문하고
- 현재의 내 위치가 출발 위치랑 동일할 때

Recursion Case:

- 아직 방문하지 않았고, 간선이 연결 되어있는 정점을 향해 탐색

Base Case에 도달했을 때
도시를 순회하는데 든 비용을 기록, 최솟값과 대조

Base Case
- 모든 도시(정점)을 모두 방문하고
- 현재의 내 위치가 출발 위치랑 동일할 때

Recursion Case:
- 아직 방문하지 않았고, 간선이 연결 되어있는 정점을 향해 탐색


```
public static int[][] w;
public static boolean[] visited;
public static int answer = Integer.MAX_VALUE;

public static void travel(int start, int node, int sum, int cnt) {
    if(cnt == n && start == node) {
        answer = Math.min(answer, sum);
        return;
    }
    for(int i = 0; i < n; i++) {
        if(!visited[i] && w[node][i] != 0) {
            visited[i] = true;
            travel(start, i, sum + w[node][i], cnt + 1);
            visited[i] = false;
        }
    }
}
```

Base Case

Recursive Case

Ch05. 재귀 #3

6. [14888] 연산자 끼워넣기

BOJ14888: 연산자 끼워넣기

문제 요약

- 수열 사이에 덧셈(+), 뺄셈(-), 곱셈(x), 나눗셈(/) 을 넣을 수 있음
- 연산자 우선순위를 무시함 (무조건 앞 → 뒤 순서로 계산)
- 나눗셈은 몫만 취함
  - 음수의 나눗셈은 C++14의 기준을 따름 (JAVA 8/11/17 동일)
- 만들어지는 결과의 최대 / 최소를 각각 구하기

BOJ14888: 연산자 끼워넣기

음수의 나눗셈?

Q. (-10 / 3) 을 정수형 변수에 넣으면 결과는?

(1) -3

(2) -4

BOJ14888: 연산자 끼워넣기

음수의 나눗셈?

Q. (-10 / 3) 을 정수형 변수에 넣으면 결과는?

(1) -3 ← C / C++ / JAVA / Go / C#/ Rust / Perl

(2) -4 ← Ruby / Python /JS / Lua

BOJ14888: 연산자 끼워넣기

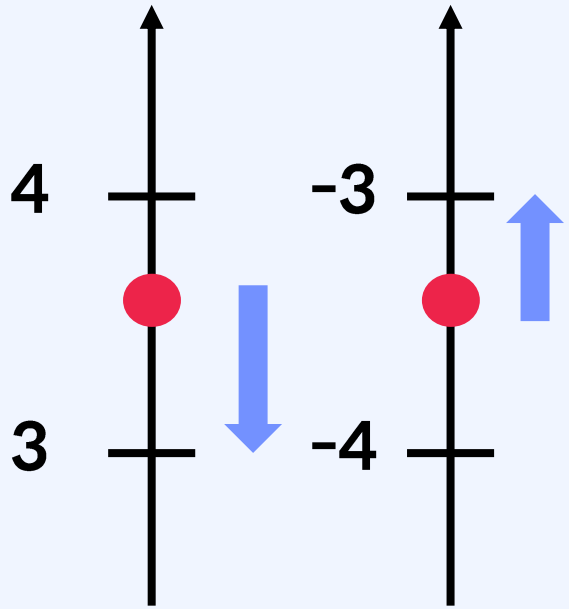
음수의 나눗셈?

Q. (-10 / 3) 을 정수형 변수에 넣으면 결과는?

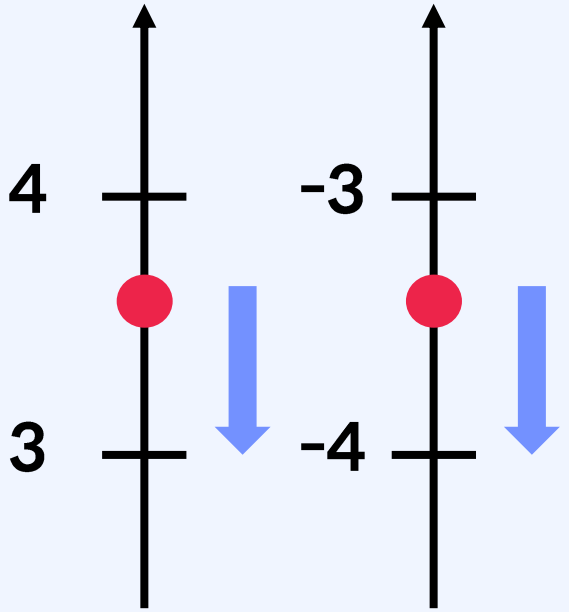
(1)-3 ← Truncate Division (버림)

(2)-4 ← Floor Division (내림)

BOJ14888: 연산자 끼워넣기



Truncate



Floor

BOJ14888: 연산자 끼워넣기

문제 분석

숫자의 개수가 최대 11개
→ 연산자의 개수는 최대 10개

숫자의 순서가 바뀌지 않고, 연산자만 바뀜
경우의 수는 총 4^10 == 2^20 == 1,048,576

연산자별 개수 제한이 있으므로, 실제로는 경우의 수가 더 적음

BOJ14888: 연산자 끼워넣기

Base Case
- 수열의 마지막 숫자에 대한 연산이 완료되었을 때
  - 결과 값이 max를 갱신 가능하다면 대입
  - 결과 값이 min을 갱신 가능하다면 대입

Recursive Case
- 사칙연산
  - {index} 번째의 수를 앞서 구한 값에 덧셈
  - {index} 번째의 수를 앞서 구한 값에 뺄셈
  - {index} 번째의 수를 앞서 구한 값에 곱셈
  - {index} 번째의 수를 앞서 구한 값에 나눗셈
- 단, 문제에서 정의한 연산 가능횟수가 남아있는 경우

BOJ14888: 연산자 끼워넣기

Base Case
- 수열의 마지막 숫자에 대한 연산이 완료되었을 때
  - 결과 값이 max를 갱신 가능하다면 대입
  - 결과 값이 min을 갱신 가능하다면 대입

```
public static void solve(int index, int sum) {
    if(index == n) {
        if(sum > max) max = sum;
        if(sum < min) min = sum;
        return;
    }
    // TODO: recursive case
}
```

Recursive Case
- 사칙연산
  - {index} 번째의 수를 앞서 구한 값에 덧셈
  - {index} 번째의 수를 앞서 구한 값에 뺄셈
  - {index} 번째의 수를 앞서 구한 값에 곱셈
  - {index} 번째의 수를 앞서 구한 값에 나눗셈
- 단, 문제에서 정의한 연산 가능횟수가 남아있는 경우


```
for (int i = 0; i < 4; i++) {
    if(operators[i] > 0) {
        operators[i]--; // 연산 가능 횟수 차감
        switch (i) {
            case PLUS -> solve(index + 1, sum + numbers[index]);
            case MINUS -> solve(index + 1, sum - numbers[index]);
            case MUL -> solve(index + 1, sum * numbers[index]);
            case DIV -> solve(index + 1, sum / numbers[index]);
        }
        operators[i]++; // 연산 가능 횟수 복구
    }
}
```

Ch05. 재귀 #3

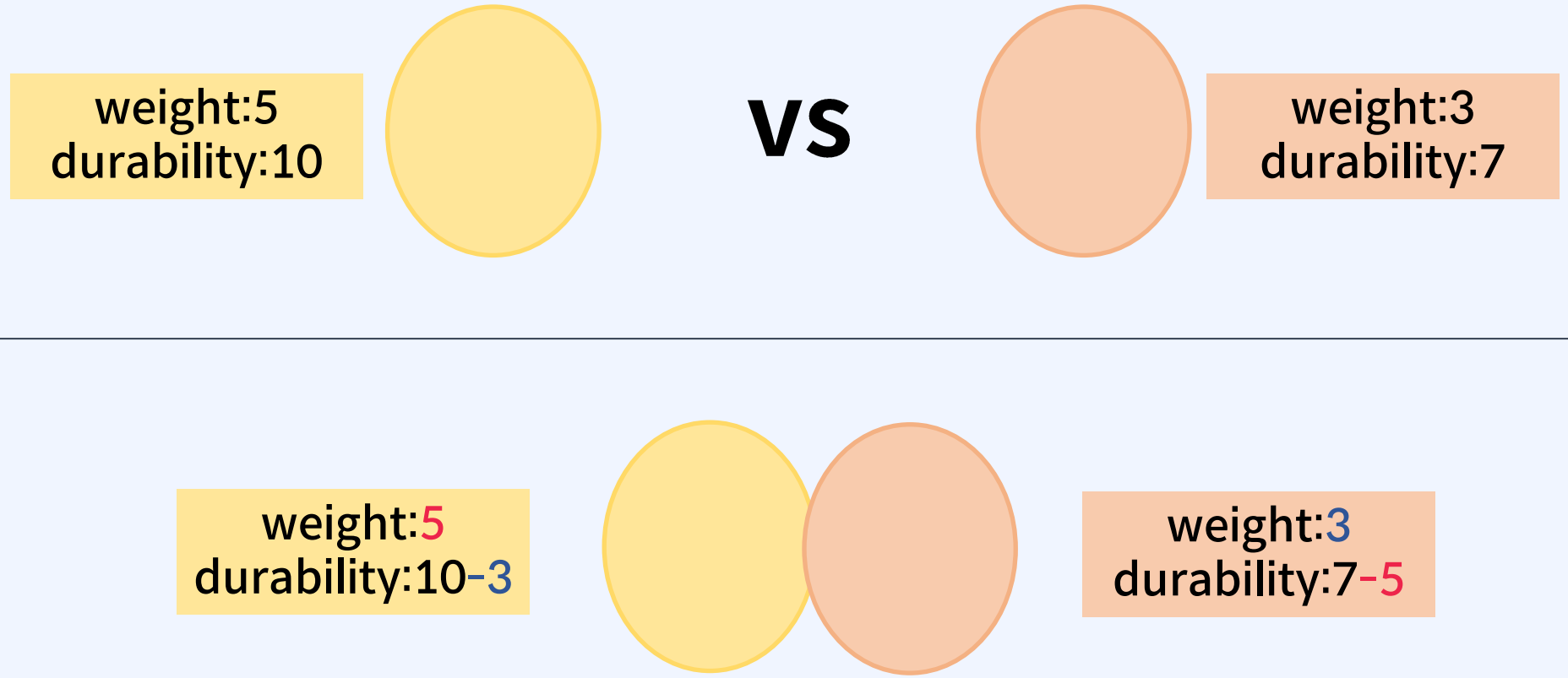
7. [16987] 계란으로 계란치기

BOJ16987: 계란으로 계란치기

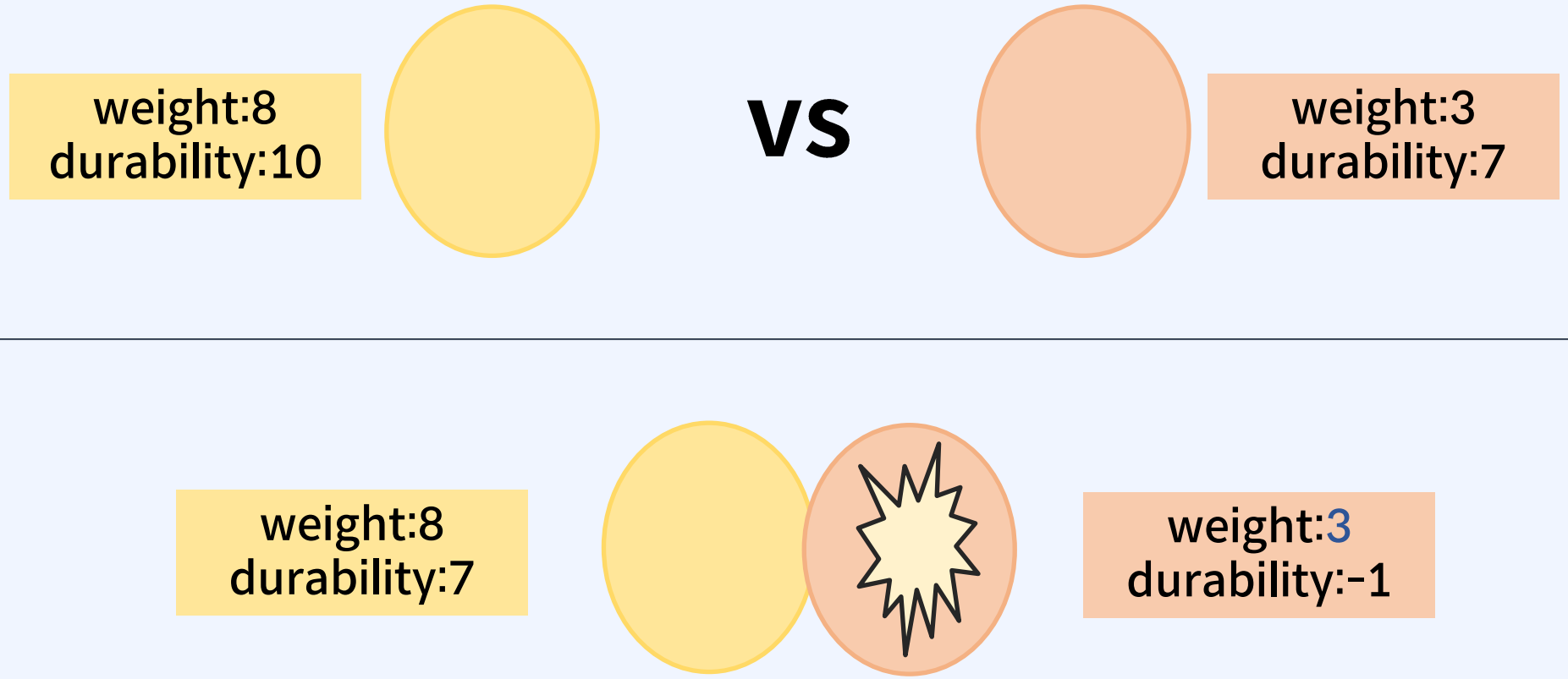
문제 요약

- 모든 계란은 무게(weight)와 내구도(durability) 상태를 보유하고 있다
- 계란 {A, B} 두개가 부딪치면 A의 내구도는 B의 무게만큼 차감된다
  동일하게 B의 내구도는 A의 무게만큼 차감된다
- 내구도가 0 이하로 낮아지면 계란은 깨진다
- 손에 들 계란은 왼쪽에서 오른쪽 순으로 선택하고
  남아있는 임의의 계란과 충돌을 시도한다
- 가장 최근에 잡은 계란을 내려놓고, 한 칸 오른쪽 계란을 들어 위의 과정을 반복한다

BOJ16987: 계란으로 계란치기



BOJ16987: 계란으로 계란치기



BOJ16987: 계란으로 계란치기

계란의 상태관리

- 여러 개의 상태 값이 존재하고 수정 가능하다
- 데이터를 관리하는 목적의 클래스를 만들면 유지보수가 편리하다

weight: 8
durability: 10


```
class Egg {
    int durability;
    int weight;

    public Egg(int durability, int weight) {
        this.durability = durability;
        this.weight = weight;
    }
```

```
    public void fight(Egg other) {
        this.durability -= other.weight;
        other.durability -= this.weight;
    }

    public void restore(Egg other) {
        this.durability += other.weight;
        other.durability += this.weight;
    }
}
```

BOJ16987: 계란으로 계란치기

Base Case

- 손에 집을 계란이 더 이상 없는 경우
  (최근에 가장 오른쪽 계란을 잡음)
  - 깨진 계란의 수가 몇 개인지 함께 체크한다 (최대 값 갱신)

Recursive Case

- 손에 집은 계란이 아직 깨지지 않은 경우
  - [0:n) 번째 계란을 대상으로 손에 집은 계란과 충돌시킨다
  - 충돌시킬 계란이 없다면 (모두 깨짐) 오른쪽 계란을 집는다
- 손에 집은 계란이 이미 깨진 경우
  - 오른쪽 계란을 집는다

BOJ16987: 계란으로 계란치기

Base Case

- 손에 집을 계란이 더 이상 없는 경우 (최근에 가장 오른쪽 계란을 잡음)
  - 깨진 계란의 수가 몇 개인지 함께 체크한다 (최대 값 갱신)

```
if(pick == n) {
    int count = 0;
    for(int i = 0; i < n; i++) {
        if(eggs[i].durability <= 0) count++;
    }
    answer = Math.max(answer, count);
    return;
}
```

BOJ16987: 계란으로 계란치기

Recursive Case

- 손에 집은 계란이 아직 깨지지 않은 경우
  - [0:n) 번째 계란을 대상으로 손에 집은 계란과 충돌시킨다
  - 충돌시킬 계란이 없다면 (모두 깨짐) 오른쪽 계란을 집는다
- 손에 집은 계란이 이미 깨진 경우
  - 오른쪽 계란을 집는다

```
if(eggs[pick].durability > 0) {
    boolean targetExists = false;
    for(int target = 0; target < n; target++) {
        // TODO: [pick] vs [target]  계란 충돌
    }
    if(!targetExists) solve(pick + 1);
}
else {
    solve(pick + 1);
}
```

BOJ16987: 계란으로 계란치기

Recursive Case

- 손에 집은 계란이 아직 깨지지 않은 경우
  - [0:n) 번째 계란을 대상으로 손에 집은 계란과 충돌시킨다
  - 충돌시킬 계란이 없다면 (모두 깨짐) 오른쪽 계란을 집는다
- 손에 집은 계란이 이미 깨진 경우
  - 오른쪽 계란을 집는다

```
for(int target = 0; target < n; target++) {
    if(target == pick) continue;
    if(eggs[target].durability > 0) {
        targetExists = true;
        eggs[pick].fight(eggs[target]);
        solve(pick + 1);
        eggs[pick].restore(eggs[target]);
    }
}
```