🚀 [스마트 플밍 4기] 1주차 알고리즘 스터디 회고

📅 금주 목표 & 현황

- [ ] **쉬움**: (주사위 게임 1) - 성공
- [ ] **중간**: (주사위의 개수) - 성공
- [ ] **도전**: (주사위 게임 2) - 성공

💡 문제 1: [주사위 게임 1]

> **출처**: https://school.programmers.co.kr/learn/courses/30/lessons/181839
> **난이도**: ★
> **유형**: 조건문 활용

1. 🧐 접근 방식 (How I Thought)

(규칙 2.1.1: 코드 설명보다는 '왜' 이 방식을 택했는지 기록)

- **문제 분석**: 둘 중 하나만 홀수 인 경우는 `||` 연산으로 하였을 때 둘 다 홀수인 경우도 포함할 수 있기 때문에 else 에 포함시켰다.
- **알고리즘 선택**:
  ① 가장 먼저 숫자의 성질(홀수/짝수)을 파악해야 합니다. 이를 위해 `나머지 연산자(%)` 를 사용했습니다.
  ② 두 수 a와 b의 상태 조합(Case)에 따라 다른 연산을 수행해야 하므로 `if-else` 문을 사용했습니다.
  ③ 수학적으로는 a < b일 때 -1을 곱해주는 방식이지만, 프로그래밍에서는 라이브러리 함수인 `Math.Abs()`를 사용하여 처리했습니다.
- **자료구조**: 기본 자료형만 사용되었습니다.
  - `int` (정수형)
    - 용도: 주사위의 눈금(a, b)과 계산된 **결과값(점수)** 을 저장하는 데 사용되었습니다.
    - 이유: 문제에서 주어지는 숫자가 정수이고, 반환해야 할 점수도 정수이기 때문입니다.
  - `bool` (논리형)
    - 용도: 해당 숫자가 홀수인지 짝수인지 판별한 결과`(true / false)`를 저장하는 데 사용되었습니다. `(isAOdd, isBOdd)`
    - 이유: 조건문(if)에서 판단의 근거로 사용하기 위해 참/거짓 값을 저장할 필요가 있었습니다.

2. 🚧 트러블 슈팅 (Retrospective)

(규칙 3.1: 실패와 시행착오 공유)

- **막혔던 점**: 처음에는 둘 중 하나만 홀수인 경우를 `(a % 2 != 0) || (b % 2 != 0)`으로 조건을 설정하는 논리적 오류를 범했다.
- **해결**: 조건식이 명확히 보일 수 있도록 bool 형 변수 `isAOdd`, `isBOdd` 로 치환하여 조건식을 재설정함.

3. 💻 나의 풀이 코드 (C#)

▶ 코드 보기 (Click)

```
using System;

public class Solution
{
    public int solution(int a, int b)
    {
        bool isAOdd = (a % 2 != 0);
        bool isBOdd = (b % 2 != 0);

        // 1. a와 b가 모두 홀수인 경우
        if (isAOdd && isBOdd) {
            return (a * a) + (b * b);
        }
        // 2. a와 b가 모두 홀수가 아닌 경우 (즉, 모두 짝수인 경우)
        else if (!isAOdd && !isBOdd) {
            return Math.Abs(a - b);
        }
        // 3. a와 b 중 하나만 홀수인 경우
        else {
            return 2 * (a + b);
        }
    }
}
```

💡 문제 2: [주사위의 개수]

> **출처**: https://school.programmers.co.kr/learn/courses/30/lessons/120845 **난이도**: ★★ **유형**: 조건문 활용

1. 🧐 접근 방식 (How I Thought)

(규칙 2.1.1: 코드 설명보다는 '왜' 이 방식을 택했는지 기록)

- **문제 분석**:
  - 시간 복잡도: O(1) (상수 시간)

    입력된 상자의 크기가 아무리 커져도(예: 100억 미터), 계산은 딱 3번의 나눗셈과 2번의 곱셈으로 끝납니다. 반복문이 없으므로 속도가 가장 빠릅니다.
  - 공간 복잡도: O(1) (상수 공간)

    추가적인 배열이나 리스트를 만들지 않고, 계산 결과만 바로 반환하므로 메모리를 거의 쓰지 않는 아주 효율적인 코드입니다.
- **알고리즘 선택**:
  ① 정수 나눗셈 (Integer Division) 알고리즘 컴퓨터가 정수끼리 나눗셈을 할 때 소수점을 버림 처리하는

특성을 이용했습니다.

현실의 논리: "상자 가로가 10cm이고 주사위가 3cm면, 3.33개가 들어가니 실제로는 3개만 넣을 수 있다."

코드의 논리: 10 / 3의 결과는 정수 3이다. (나머지 버림)

이 나눗셈 연산 자체가 '물리적으로 주사위를 채워 넣는 과정'을 시뮬레이션하는 알고리즘 역할을 대신한 것입니다.

② 부피 계산의 응용 (기하학적 사고) 단순히 전체 부피 나누기 주사위 부피를 하지 않고, 각 모서리별로 들어가는 개수를 먼저 구해서 곱한다는 순서를 짠 것 자체가 이 문제의 핵심 알고리즘입니다.

- **자료구조**:
  - **int[] box (정수형 배열)**입니다.
    - 역할: 가로, 세로, 높이(3개의 데이터)를 따로따로 변수 3개(w, l, h)에 담지 않고, box라는 하나의 이름으로 묶어서 관리했습니다.
    - 특징: box[0], box[1], box[2]처럼 **인덱스(순서)** 를 통해 데이터에 아주 빠르게 접근했습니다. 데이터가 메모리상에 나란히 붙어 있어 컴퓨터가 읽기 편한 구조입니다.

2. 🚧 트러블 슈팅 (Retrospective)

(규칙 3.1: 실패와 시행착오 공유)

- **막혔던 점**: 처음에는 부피를 주사위 부피를 나누어서 계산하려고 했음. 코드가 한눈에 들어오지 않았음.
- **해결**:

```
int widthCount = box[0] / n;
int lengthCount = box[1] / n;
int heightCount = box[2] / n;
```

3. 💻 나의 풀이 코드 (C#)

▶ 코드 보기 (Click)

```
using System;

public class Solution {
    public int solution(int[] box, int n) {
        int answer = 1;
        // 1. 가로, 세로, 높이에 각각 주사위가 몇 개 들어가는지 계산 (정수 나눗셈)
        int widthCount = box[0] / n;
        int lengthCount = box[1] / n;
        int heightCount = box[2] / n;

        // 2. 각 모서리에 들어가는 개수를 곱하여 총 개수 반환
        return answer = widthCount * lengthCount * heightCount;
    }
}
```

💡 문제 3: [주사위 게임 2]

> **출처**: https://school.programmers.co.kr/learn/courses/30/lessons/181930 **난이도**: ⭐⭐⭐ **유형**: 조건문 활용

1. 🧐 접근 방식 (How I Thought)

(규칙 2.1.1: 코드 설명보다는 '왜' 이 방식을 택했는지 기록)

- **문제 분석**:
- (1) 가독성 (사람이 읽기 편한 코드)
  - 코드는 기계가 실행하지만, 사람이 읽고 수정합니다. `if (a == b && b == c)` 이 코드는 누가 봐도 `"셋 다 같으면"`이라고 해석됩니다.
  - 현업에서는 협업이 중요하므로, 직관적인 로직을 선호하여 이렇게 작성했습니다.
- (2) 연산 성능 최적화 (Math.Pow 미사용)
  - 정수 연산 사용: a * a와 같이 단순 정수 곱셈을 사용하여 컴퓨터가 가장 빠르게 처리할 수 있는 방식을 택했습니다.
  - 불필요한 형변환 방지: double에서 int로 변환하는 과정을 없애 오버헤드를 줄였습니다.
- (3) 유지보수의 유연성
  - 만약 문제 조건이 갑자기 이렇게 바뀐다면 어떨까요? "세 숫자가 모두 같으면 점수를 0점으로 한다." 첫 번째 `if`문 내부만 `return 0;`으로 바꾸면 끝입니다.
- **알고리즘 선택**:
- 조건 분기 (Top-down Logic):
  - 가장 까다로운 조건부터 순차적으로 걸러내는 방식을 사용했습니다.
  - 가장 좁은 조건: 세 개가 모두 같은가? (A && B)
  - 그다음 조건: (위가 아니라면) 두 개라도 같은가? (A || B)
  - 나머지: 모두 다른가? (else) 이렇게 순서를 배치하면, 두 번째 else if에서 "세 개가 모두 같은 경우"를 굳이 다시 검사할 필요가 없어 논리가 단순해집니다.
- 선행 계산 (Pre-computation):
  - sum1, sum2, sum3를 조건문 진입 전에 미리 계산했습니다.
  - 어떤 조건에 걸리든 sum1은 무조건 필요하고, sum2, sum3도 수식이 복잡하므로 미리 변수에 담아두어 코드 중복을 방지하고 가독성을 높였습니다.
- **자료구조**: **기본 정수형(int)** 만을 사용했습니다.

2. 🚧 트러블 슈팅 (Retrospective)

(규칙 3.1: 실패와 시행착오 공유)

- **막혔던 점**: 없음

- **해결**: 없음

3. 💻 나의 풀이 코드 (C#)

▶ 코드 보기 (Click)

```
using System;

public class Solution
{
    public int solution(int a, int b, int c)
    {
        int answer = 0;

        // 계산 편의를 위해 각 차수의 합을 미리 변수에 저장
        int sum1 = a + b + c;
        int sum2 = (a * a) + (b * b) + (c * c);
        int sum3 = (a * a * a) + (b * b * b) + (c * c * c);

        // 1. 세 숫자가 모두 같은 경우
        if (a == b && b == c)
        {
            answer = sum1 * sum2 * sum3;
        }
        // 2. 세 숫자 중 어느 두 숫자는 같고 나머지 하나는 다른 경우
        // (위의 if문이 거짓일 때 실행되므로, '세 숫자가 모두 같은 경우'는 자연스럽게
제외됨)
        else if (a == b || a == c || b == c)
        {
            answer = sum1 * sum2;
        }
        // 3. 세 숫자가 모두 다른 경우
        else
        {
            answer = sum1;
        }

        return answer;
    }
}
```