Chapter 05.
분석과 구현

Clip 01 | [1917] 정육면체 전개도
정육면체의 상태관리

Clip 02 | [1525] 퍼즐
데이터 가공과 탐색

Clip 03 | [16939] 2×2×2 큐브
복잡한 구현

Clip 04 | [14599] 인공지능 테트리스 (Large)
복잡한 구현과 예외 케이스 처리

Ch05. 분석과 구현

1. [1917] 정육면체 전개도

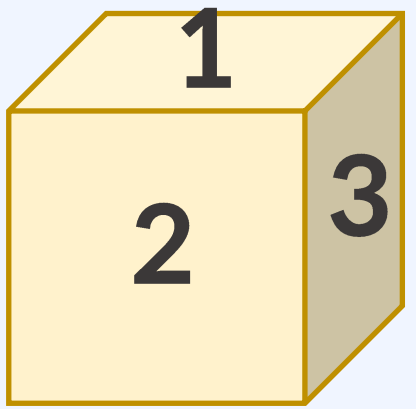
BOJ1917: 정육면체 전개도

문제 요약

- 3개의 입력 데이터
- 6*6 공간 안에 0 / 1로 입력이 주어짐
  - 1로 표현된 위치가 정사각형
- 정사각형의 집합이 정육면체의 전개도가 될 수 있는지?
  - 단, 정사각형들은 서로 인접하게 주어짐

BOJ1917: 정육면체 전개도

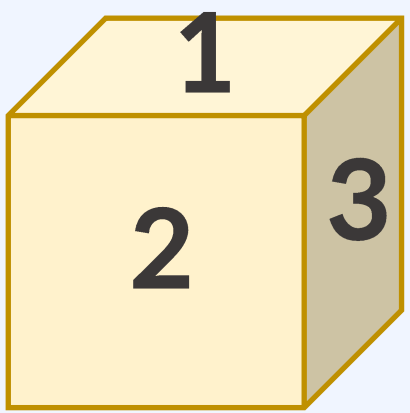
문제 분석

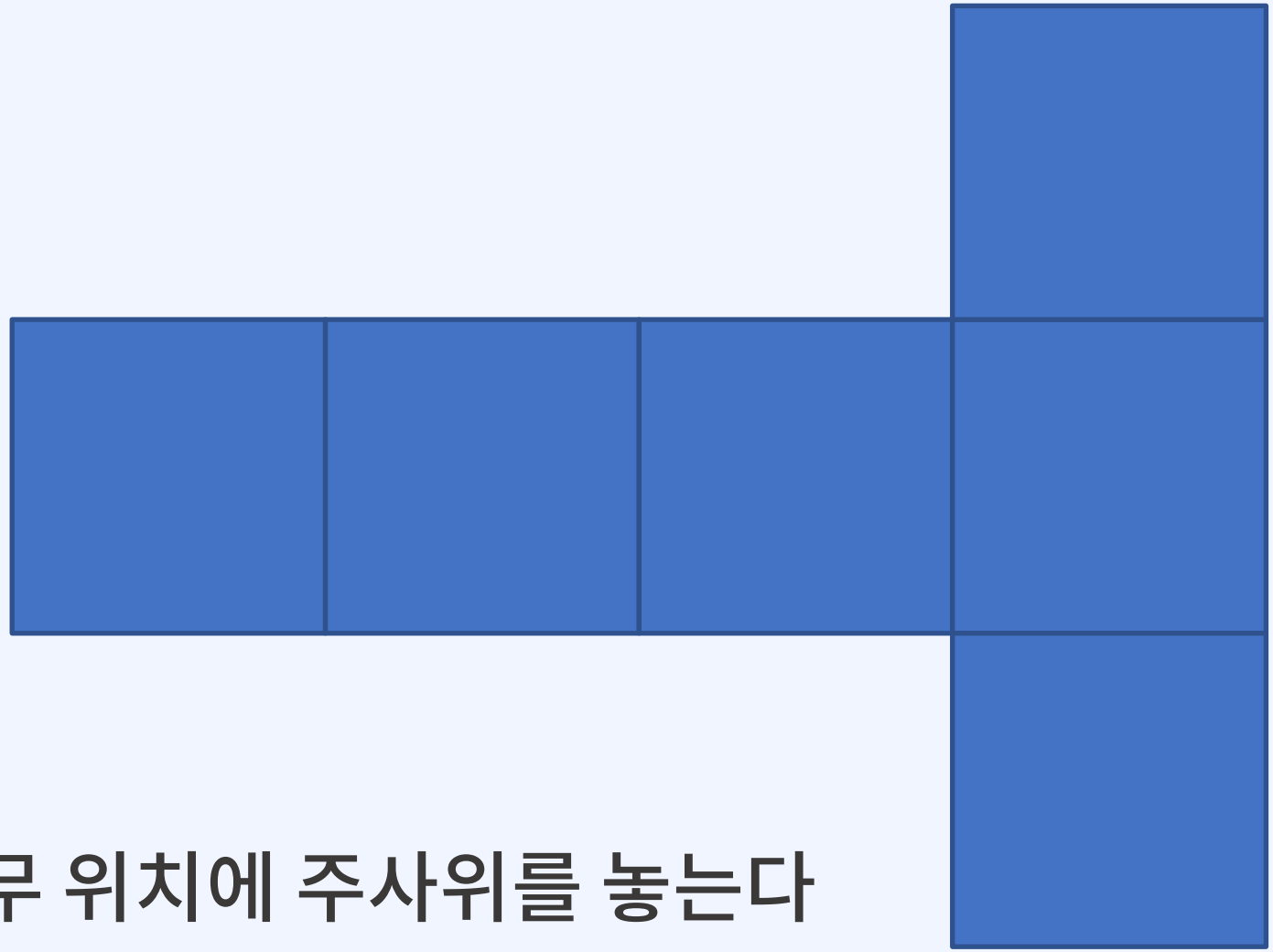


- 다양한 푸는 방법이 나올 수 있다
- 아이디어: 전개도 위에서 주사위를 굴렸을 때
  정상적인 전개도라면? 1~6이 한번씩 등장한다
- tip) 주사위는 맞은편과의 숫자의 합이 7이다

BOJ1917: 정육면체 전개도

문제 분석

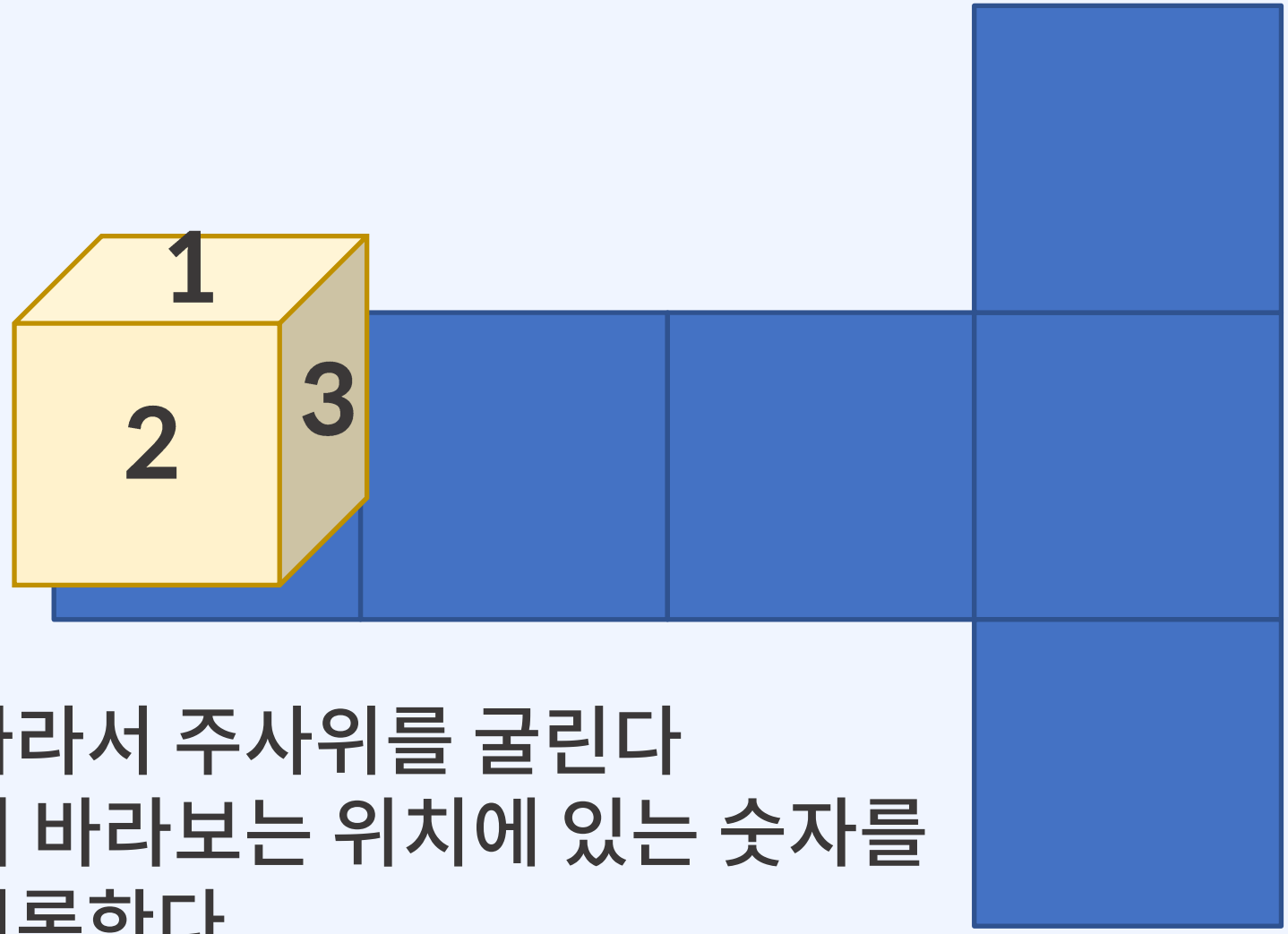




전개도의 아무 위치에 주사위를 놓는다

BOJ1917: 정육면체 전개도

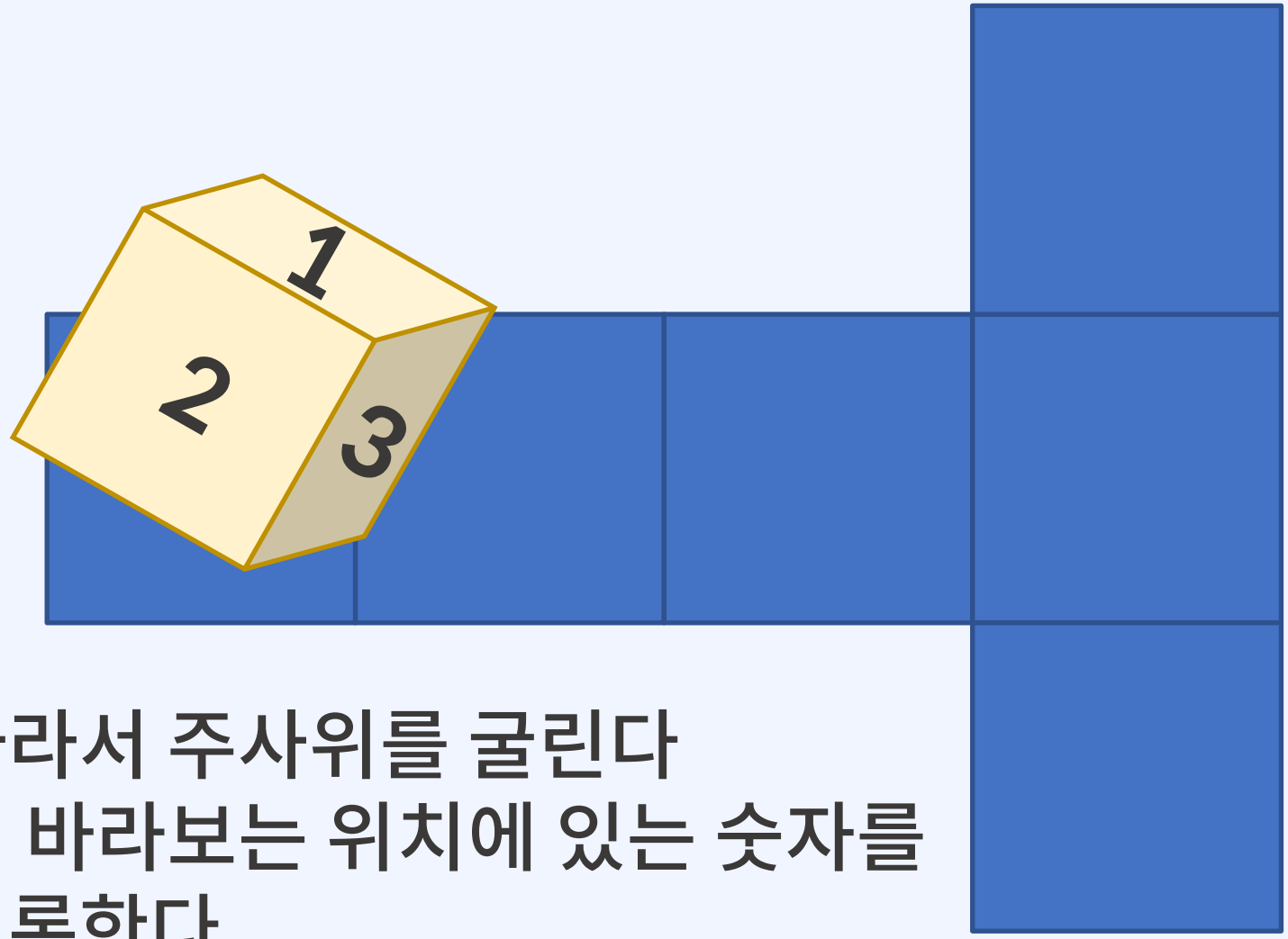
문제 분석



전개도를 따라서 주사위를 굴린다
이때 위에서 바라보는 위치에 있는 숫자를
전개도에 기록한다

BOJ1917: 정육면체 전개도

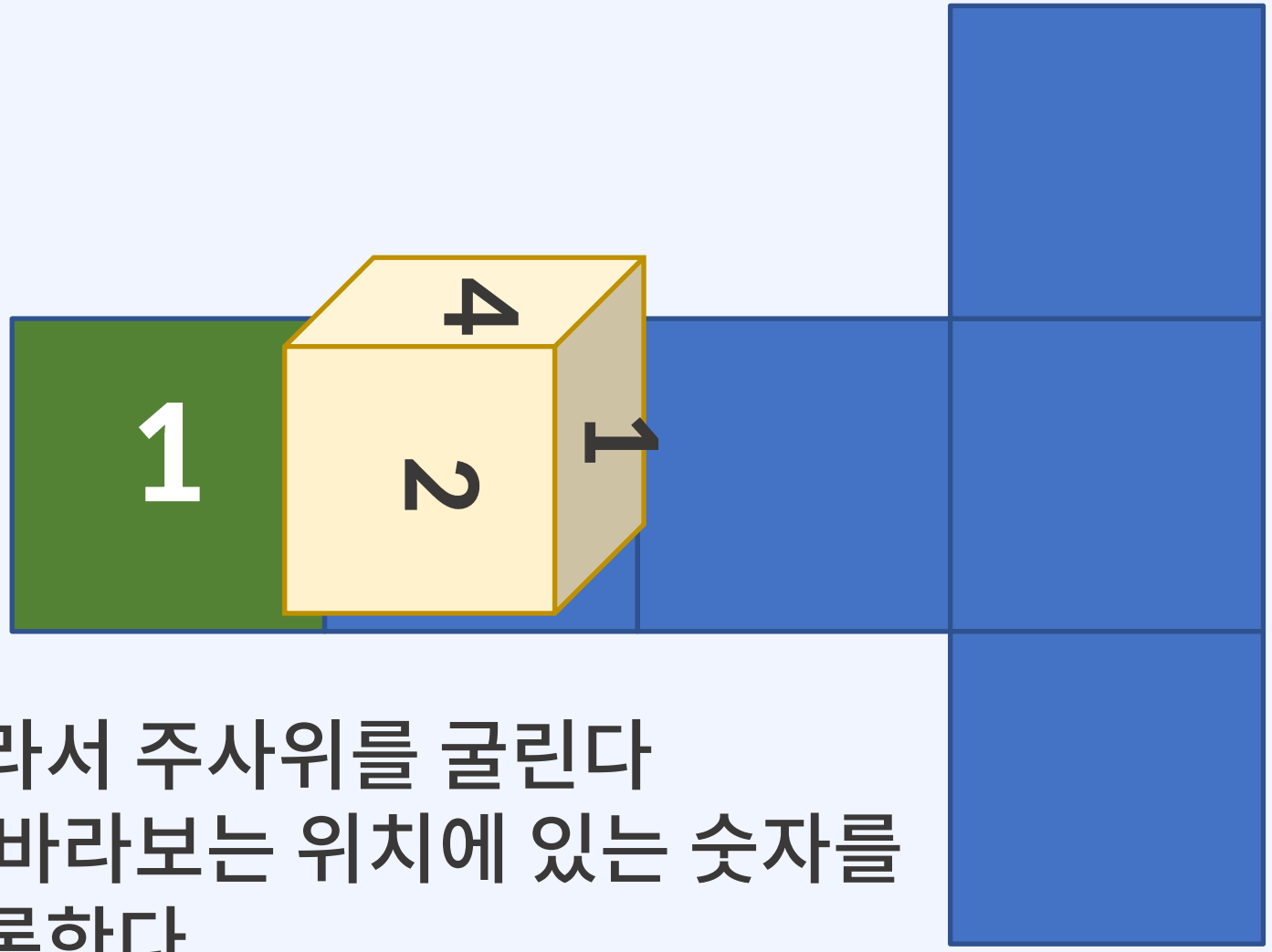
문제 분석



전개도를 따라서 주사위를 굴린다
이때 위에서 바라보는 위치에 있는 숫자를
전개도에 기록한다

BOJ1917: 정육면체 전개도

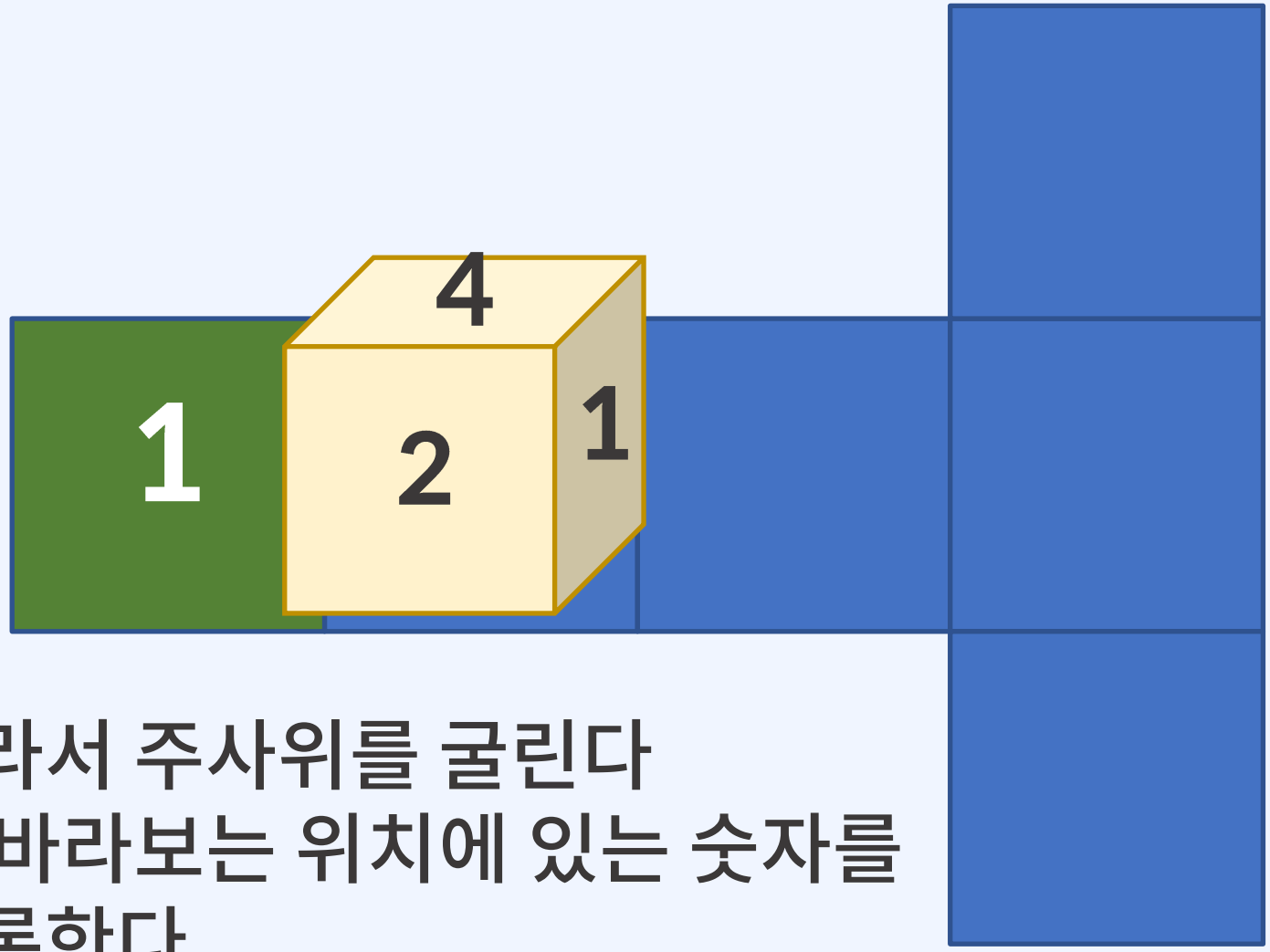
문제 분석



전개도를 따라서 주사위를 굴린다
이때 위에서 바라보는 위치에 있는 숫자를
전개도에 기록한다

BOJ1917: 정육면체 전개도

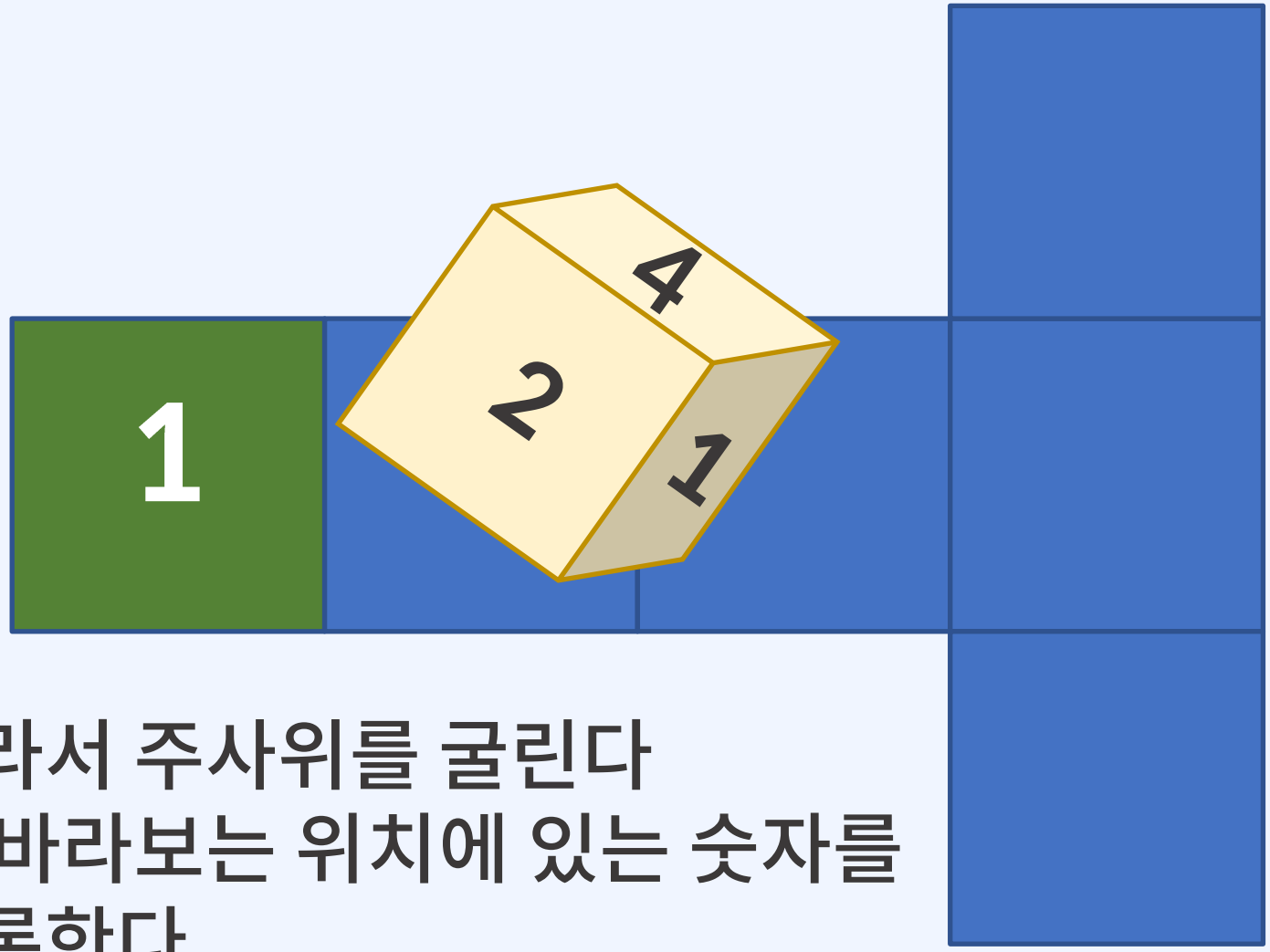
문제 분석



전개도를 따라서 주사위를 굴린다
이때 위에서 바라보는 위치에 있는 숫자를
전개도에 기록한다

BOJ1917: 정육면체 전개도

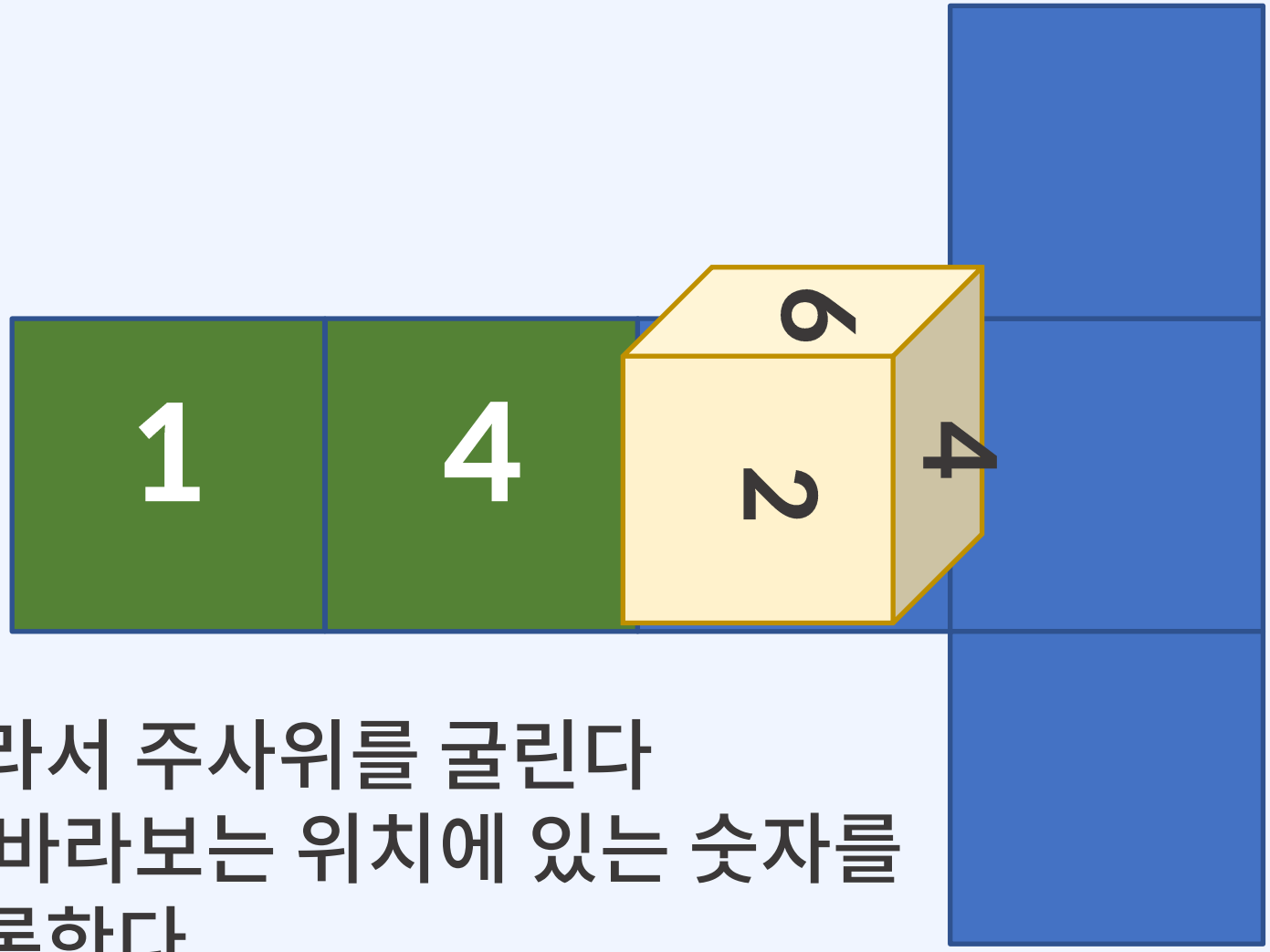
문제 분석



전개도를 따라서 주사위를 굴린다
이때 위에서 바라보는 위치에 있는 숫자를
전개도에 기록한다

BOJ1917: 정육면체 전개도

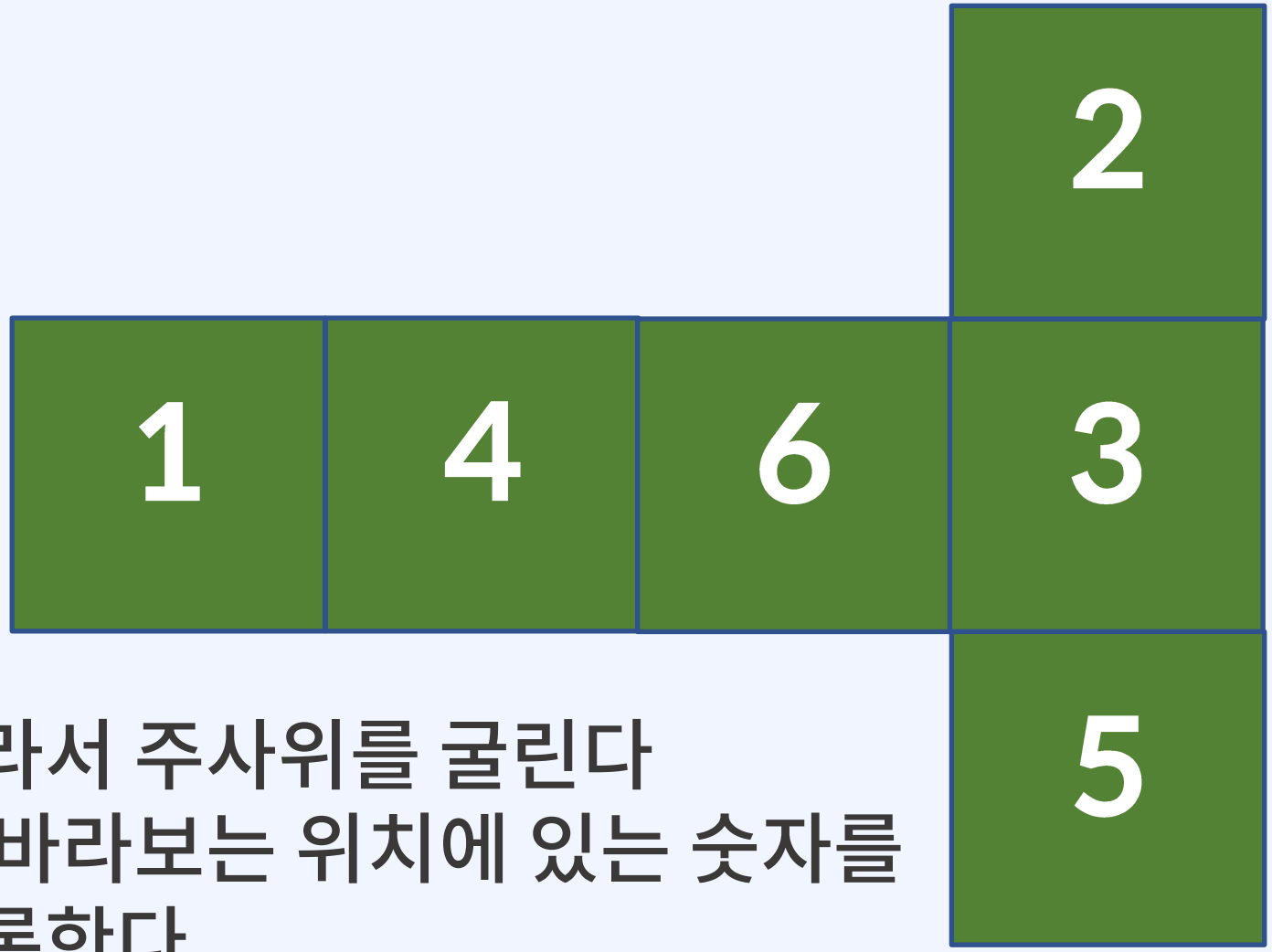
문제 분석



전개도를 따라서 주사위를 굴린다
이때 위에서 바라보는 위치에 있는 숫자를
전개도에 기록한다

BOJ1917: 정육면체 전개도

문제 분석



전개도를 따라서 주사위를 굴린다
이때 위에서 바라보는 위치에 있는 숫자를
전개도에 기록한다

BOJ1917: 정육면체 전개도

문제 분석

- 아이디어: 전개도 위에서 주사위를 굴렸을 때
  정상적인 전개도라면? 1~6이 한번씩 등장한다

- 문제 조건
  - 36개의 숫자 중 1은 정확히 6개가 있다
  - 정사각형들이 서로 떨어져 있는 경우는 없다.

BOJ1917: 정육면체 전개도

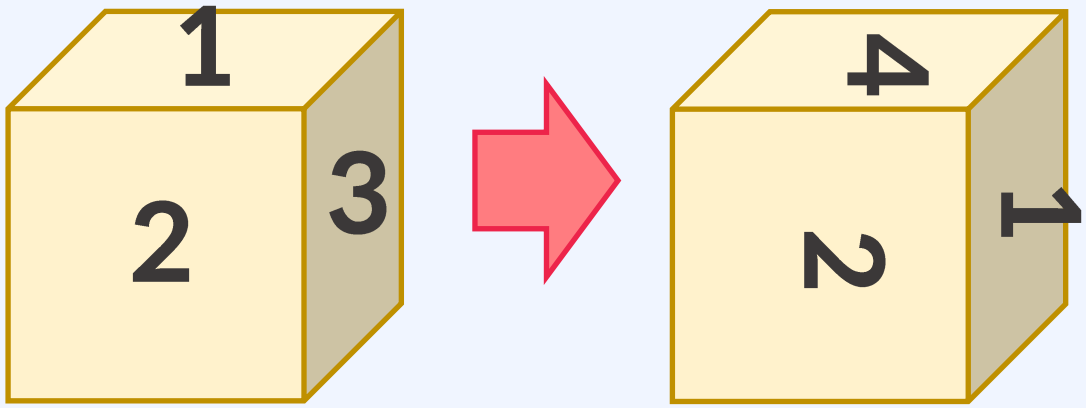
문제 분석

- 주사위를 관리하는 객체를 만들고
- 주사위를 상/하/좌/우 로 굴리는 함수를 구현한다음
- 탐색(DFS /BFS)을 굴린다

- tip) 주사위는 맞은편과의 숫자의 합이 7이다

BOJ1917: 정육면체 전개도

구현 – 오른쪽 회전

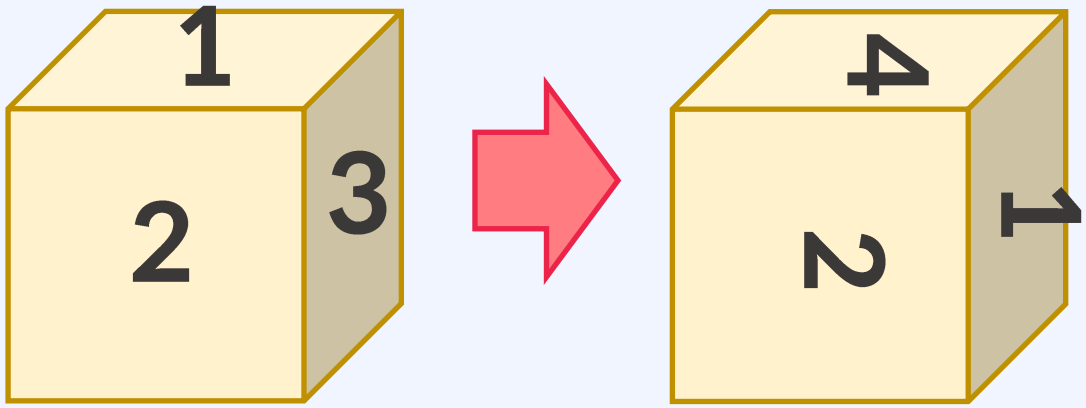


- 1의 위치에 4가 등장
- 4의 위치에 6이 등장
- 6의 위치에 3이 등장
- 3의 위치에 1이 등장

- tip) 주사위는 맞은편과의 숫자의 합이 7이다

BOJ1917: 정육면체 전개도

구현 – 오른쪽 회전

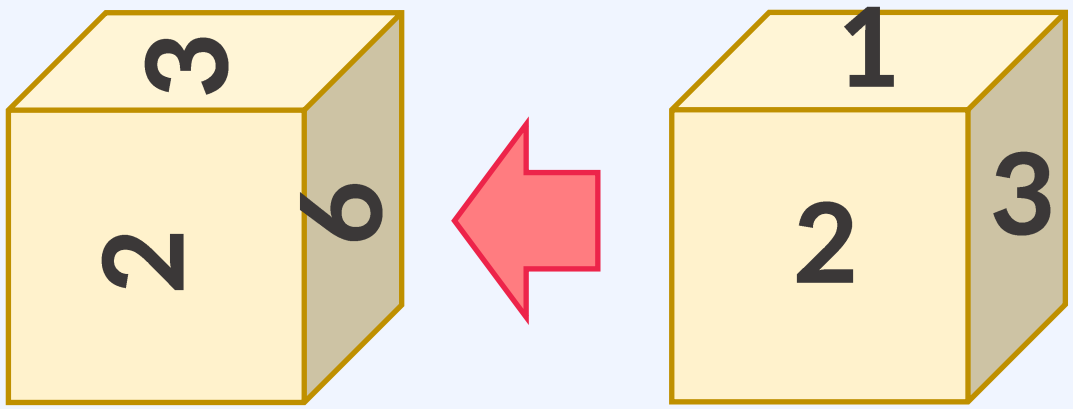



```
public void right() {
    int temp = cube[1];
    cube[1] = cube[4];
    cube[4] = cube[6];
    cube[6] = cube[3];
    cube[3] = temp;
}
```

- tip) 주사위는 맞은편과의 숫자의 합이 7이다

BOJ1917: 정육면체 전개도

구현 – 왼쪽 회전

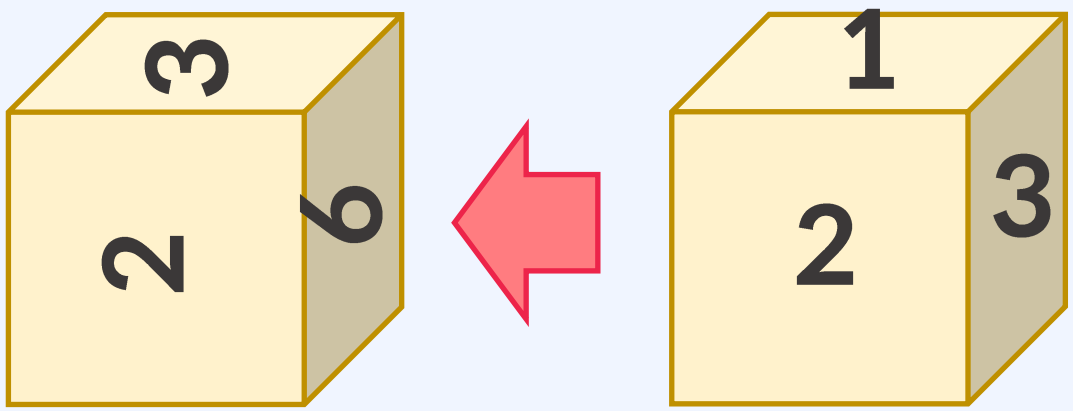


- 1의 위치에 3이 등장
- 3의 위치에 6이 등장
- 6의 위치에 4가 등장
- 4의 위치에 1이 등장

- tip) 주사위는 맞은편과의 숫자의 합이 7이다

BOJ1917: 정육면체 전개도

구현 – 왼쪽 회전

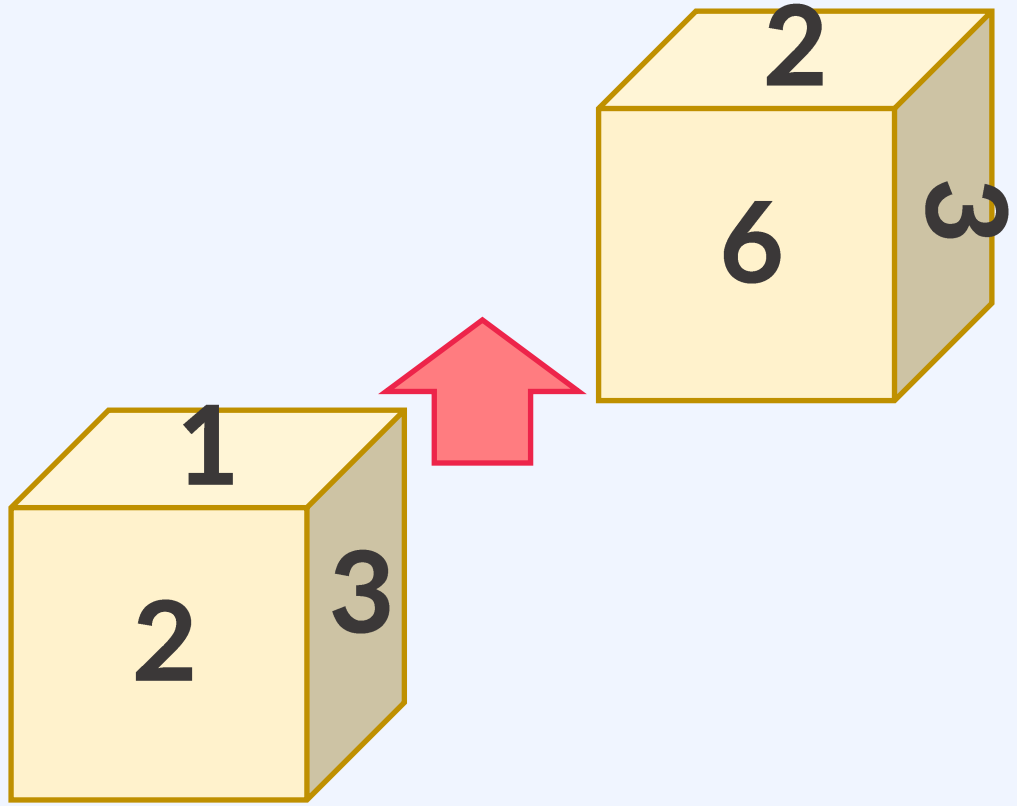



```
public void left() {
    int temp = cube[1];
    cube[1] = cube[3];
    cube[3] = cube[6];
    cube[6] = cube[4];
    cube[4] = temp;
}
```

- tip) 주사위는 맞은편과의 숫자의 합이 7이다

BOJ1917: 정육면체 전개도

구현 – 위쪽 회전

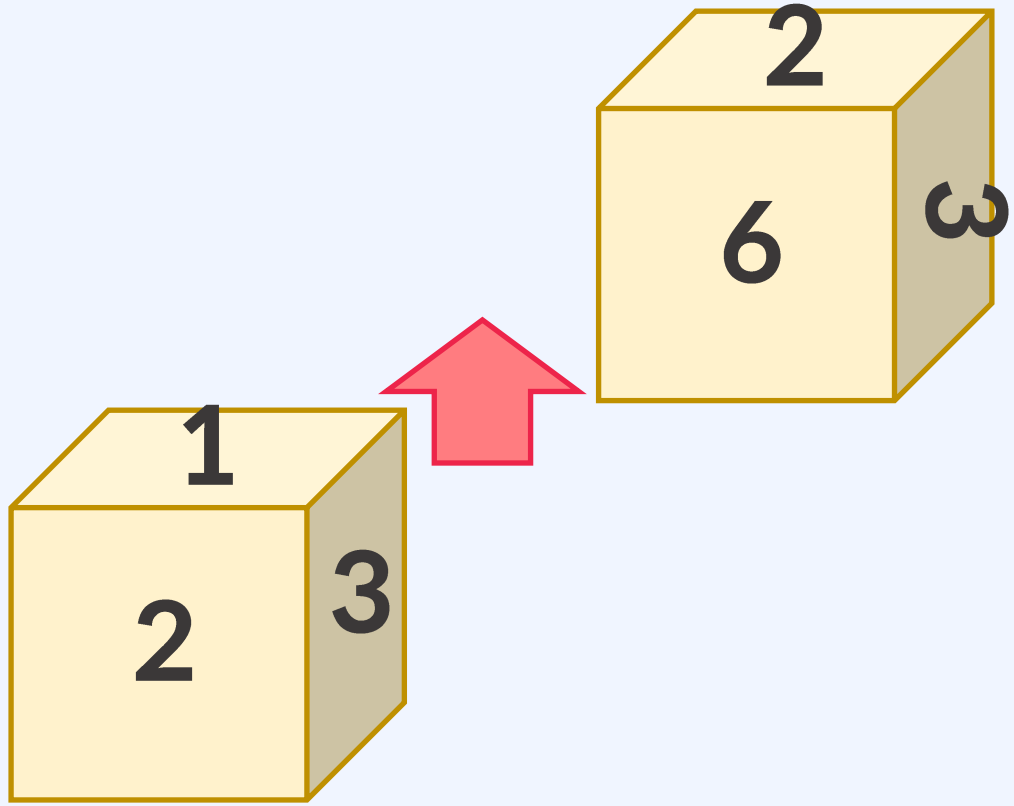


- 1의 위치에 2가 등장
- 2의 위치에 6이 등장
- 6의 위치에 5가 등장
- 5의 위치에 1이 등장

- tip) 주사위는 맞은편과의 숫자의 합이 7이다

BOJ1917: 정육면체 전개도

구현 – 위쪽 회전

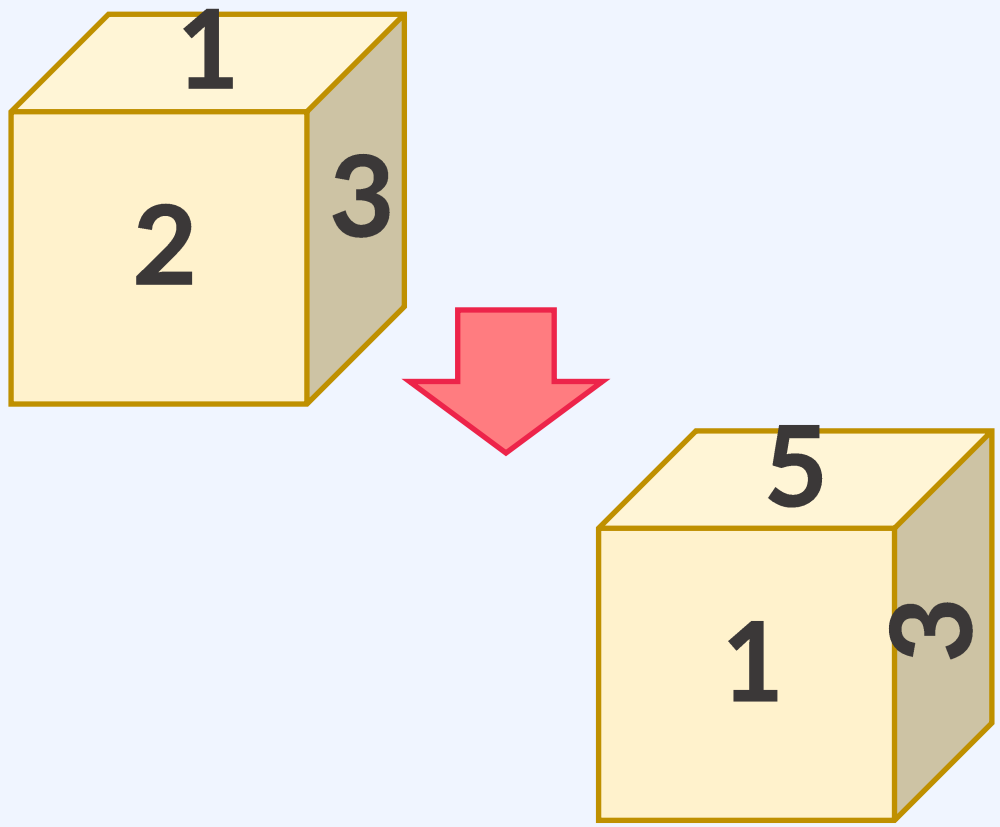



```
public void up() {
    int temp = cube[1];
    cube[1] = cube[2];
    cube[2] = cube[6];
    cube[6] = cube[5];
    cube[5] = temp;
}
```

- tip) 주사위는 맞은편과의 숫자의 합이 7이다

BOJ1917: 정육면체 전개도

구현 – 아래쪽 회전

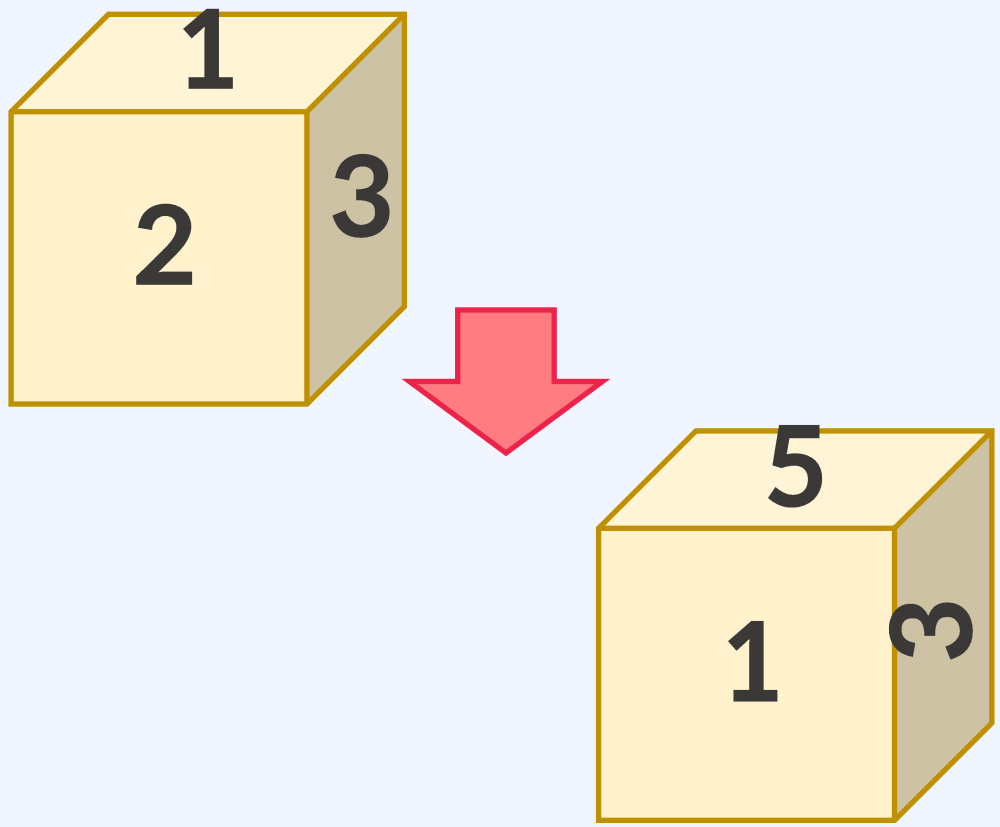


- 1의 위치에 5가 등장
- 5의 위치에 6이 등장
- 6의 위치에 2가 등장
- 2의 위치에 1이 등장

- tip) 주사위는 맞은편과의 숫자의 합이 7이다

BOJ1917: 정육면체 전개도

구현 – 아래쪽 회전




```
public void down() {
    int temp = cube[1];
    cube[1] = cube[5];
    cube[5] = cube[6];
    cube[6] = cube[2];
    cube[2] = temp;
}
```

BOJ1917: 정육면체 전개도

구현 – 생성자, 조회

```
class Dice {
    private int[] cube;
    public Dice() {
        cube = new int[7];
        for (int i = 1; i <= 6; i++) {
            cube[i] = i;
        }
    }
```

```
public Dice(int[] c) {
        cube = new int[7];
        for (int i = 1; i <= 6; i++) {
            cube[i] = c[i];
        }
    }
```

```
public int getTop() {
    return cube[1];
}

public int[] getCube() {
    return cube;
}
```

깊은 복사를 하는 생성자 오버라이딩

BOJ1917: 정육면체 전개도

구현 – 좌표 클래스, Pair

```
class Point {
    int r;
    int c;

    public Point(int r, int c) {
        this.r = r;
        this.c = c;
    }
}
```

```
class Pair<T, U> {
    T first;
    U second;

    public Pair(T first, U second) {
        this.first = first;
        this.second = second;
    }
}
```

객체 2개를 묶어서 표현할 Pair
C++ / Kotlin 의 Pair와 동일한 역할

BOJ1917: 정육면체 전개도

구현 – 초기화

```
int startR = 0, startC = 0;
for (int i = 1; i <= n; i++) {
    for (int j = 1; j <= n; j++) {
        board[i][j] = sc.nextInt();
        if (board[i][j] == 1) {
            startR = i;
            startC = j;
        }
    }
}
```

입력을 받으며 시작 좌표 탐색
1이 있는 아무 좌표에서 시작하면 된다

BOJ1917: 정육면체 전개도

구현 – 초기화

```
Dice dice = new Dice();

Stack<Pair<Point, Dice>> stack = new Stack<>();
stack.push(new Pair<>(new Point(startR, startC), dice));

diceCheck[dice.getTop()] = 1;

check[startR][startC] = 1;
```

DFS 탐색을 진행할 스택 생성
좌표 체크와 주사위 숫자를 체크할 배열 생성

```
int[] dr = {-1, 0, 1, 0};
int[] dc = {0, 1, 0, -1};
final int UP = 0;
final int RIGHT = 1;
final int DOWN = 2;
final int LEFT = 3;
```

상 / 우 / 하/ 좌
다음 탐색 방향 정의

BOJ1917: 정육면체 전개도

구현 – 탐색(DFS)

```
while (!stack.isEmpty()) {
    Pair<Point, Dice> pair = stack.pop();
    Point now = pair.first;
    Dice nowDice = pair.second;
    diceCheck[nowDice.getTop()] = 1;
```

스택에서 현재 탐색 좌표와 주사위 상태 획득
전개도의 위에서 바라본 수를 체크 배열에 기록

BOJ1917: 정육면체 전개도

구현 – 탐색(DFS)

```
for (int i = 0; i < 4; i++) {
    Point next = new Point(now.r + dr[i], now.c + dc[i]);
    if (isRange(next.r, next.c, n) && board[next.r][next.c] != 0 && check[next.r][next.c] == 0) {
        Dice nextDice = new Dice(nowDice.getCube());
        switch (i) {
            case UP -> nextDice.up();
            case RIGHT -> nextDice.right();
            case DOWN -> nextDice.down();
            case LEFT -> nextDice.left();
        }
        check[next.r][next.c] = 1;
        stack.push(new Pair<>(new Point(next.r, next.c), nextDice));
    }
}
```

다음 좌표가 범위 내인지 체크
전개도상으로 갈 수 있는 좌표인지 체크
아직 방문하지 않은 좌표인지 체크

다음 탐색 좌표가 상 / 우 / 하 / 좌 일 때
주사위를 조건에 맞게 회전

다음 좌표와 주사위 상태 기록
다음 탐색을 위해 스택에 푸시

Ch05. 분석과 구현

2. [1525] 퍼즐

BOJ1525: 퍼즐

문제 요약

- 3*3 공간에서 8개의 수가 담겨있는 퍼즐
- 빈 공간과 상/하/좌/우 인접한 수는 위치를 바꿀 수 있음
- 시작 퍼즐 형상이 주어졌을 때 아래와 같이 만들기 위한 최소 이동 횟수 출력 (불가능하면 -1)

```
123
456
780
```

BOJ1525: 퍼즐

문제 분석

- 관점을 조금 다르게 보자
  - {0} 이 상 / 하 / 좌 / 우 이동할 수 있는 상태
  - 3*3 격자판을 9개의 수가 담긴 일차원으로 생각
    - 123
      456 -> 123456780
      780
    - 9자리 수는 10억보다 작아서 Integer 변수 하나만 가지고 퍼즐 보드의 표현이 가능하다

BOJ1525: 퍼즐

1	0	3
4	2	5
7	8	6

문제 분석

BOJ1525: 퍼즐

문제 분석

- 사용 가능한 메모리가 매우 작다

- 이전 문제처럼 탐색 케이스마다 메모리에 할당하면 메모리가 초과한다
  - 재사용 가능한 객체의 인스턴스화를 막아야 한다
  - 케이스의 중복 체크를 Set을 활용한다
    - 중복 체크에 O(n log n)이 걸리지만 메모리는 적게 사용할 수 있다

BOJ1525: 퍼즐

문제 분석

int[][]

1	0	3
4	2	5
7	8	6

int

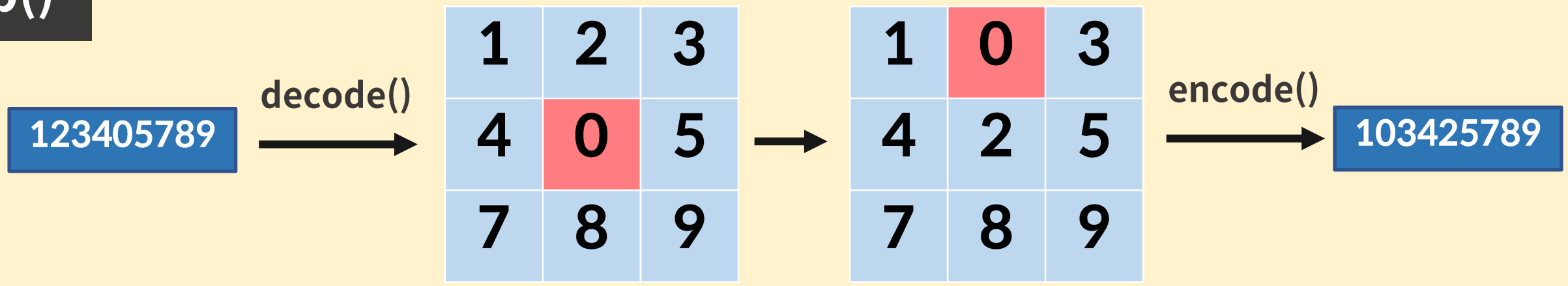
103425786

좌표 탐색을 쉽게 처리하기위해, 배열과 Integer를
자유 자재로 변경할 함수를 정의해 두자
단, 이때 공간은 한번 선언하고 재사용 해야 한다

BOJ1525: 퍼즐

문제 분석

up()

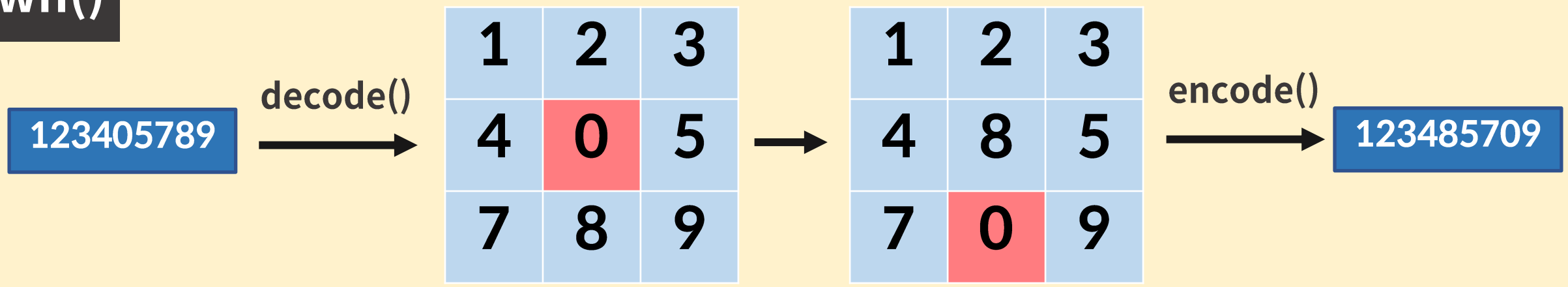


up() / down() / left() / right() 함수는
integer로 만들어진 퍼즐 판을 넣으면,
0의 위치를 바꾼 integer로 반환하도록 정의한다

BOJ1525: 퍼즐

문제 분석

down()



내부 구현은 다양한 방법으로 구현할 수 있다 (ex. 비트마스크, 모듈러 처리 등)
해당 풀이에서는 직관적으로 접근하기 위해
배열로 변환하여 처리하는 decode() / encode()로 구현한다

BOJ1525: 퍼즐

구현

```
class Puzzle {
    private static final int[][] board = new int[3][3];
    static int up(int code) {}
    static int down(int code) {}
    static int left(int code) {}
    static int right(int code) {}
}
```

0의 위치를 up/down/left/right로 옮기는 기능을 가진 클래스
내부적으로 데이터 처리를 사용할 board 배열도 정의한다

BOJ1525: 퍼즐

구현

```
private static int encode() {
    int code = 0;
    for (int i = 0; i < 3; i++) {
        for (int j = 0; j < 3; j++) {
            code = code * 10 + board[i][j];
        }
    }
    return code;
}
```

배열-> 단일 Integer 변환

```
private static int decode(int code) {
    int zero = -1;
    for (int i = 2; i >= 0; i--) {
        for (int j = 2; j >= 0; j--) {
            board[i][j] = code % 10;
            if (board[i][j] == 0)
                zero = i * 3 + j;
            code /= 10;
        }
    }
    return zero;
}
```

단일 Intger -> 배열 변환하여 저장
0의 위치도 함께 찾아서 반환

BOJ1525: 퍼즐

구현

```
static int up(int code) {
    int z = decode(code);
    if (z / 3 == 0) return -1;
    int t = board[z / 3 - 1][z % 3];
    board[z / 3 - 1][z % 3] = 0;
    board[z / 3][z % 3] = t;
    return encode();
}
```

decode()로 내부 배열에 변환 저장

움직일 수 없는 좌표면 -1 리턴

0의 위치를 위로 이동

encode()로 변환한 단일 integer 리턴

BOJ1525: 퍼즐

구현

```
class SearchData {
    int board;
    int depth;

    SearchData(int board, int depth) {
        this.board = board;
        this.depth = depth;
    }
}
```

BFS 탐색 시 board와 큐의 Depth를
표현하기 위한 클래스

BOJ1525: 퍼즐

구현

```
for (int i = 0; i < 3; i++) {
    for (int j = 0; j < 3; j++)
        board = board * 10 + sc.nextInt();
}
```

단일 Integer로 구성된 보드로 변환하여 저장

```
Set<Integer> check = new HashSet<>();
check.add(board);

Queue<SearchData> q = new LinkedList<>();
q.add(new SearchData(board, 0));
```

중복 체크는 Set 을 이용해
단일 Integer로 구성된 보드를 체크

입력 받은 데이터를 0 Depth로 BFS 시작

```
while (!q.isEmpty()) {
    SearchData data = q.poll();
    int now = data.board, depth = data.depth;
    if (now == 123456780) {
        System.out.println(depth);
        return;
    }
    int[] next = {
        Puzzle.up(now), Puzzle.down(now), Puzzle.left(now), Puzzle.right(now)
    };
    for (int i = 0; i < 4; i++) {
        if (next[i] != -1 && !check.contains(next[i])) {
            q.add(new SearchData(next[i], depth + 1));
            check.add(next[i]);
        }
    }
}
```

123456780 상태가 만들어지면
Depth를 출력하고 종료

상 / 하 / 좌 / 우 이동 시 만들어지는 상태 저장

이동 가능하고, 중복이 아닌지 체크
새 탐색의 경로로 지정하고
탐색해본 상태로 체크 Set에 저장

Ch05. 분석과 구현

3. [16939] 2x2x2 큐브

BOJ16939: 2x2x2 큐브

문제 요약

- 2 x 2 x 2 로 구성된 루빅스 큐브
- 큐브의 각 면은 양방향으로 90도 돌릴 수 있음
- 각 면의 색상이 모두 같으면, 큐브를 풀었다고 판단
- 큐브를 딱 한번 돌렸을 때, 큐브가 풀리는지 출력

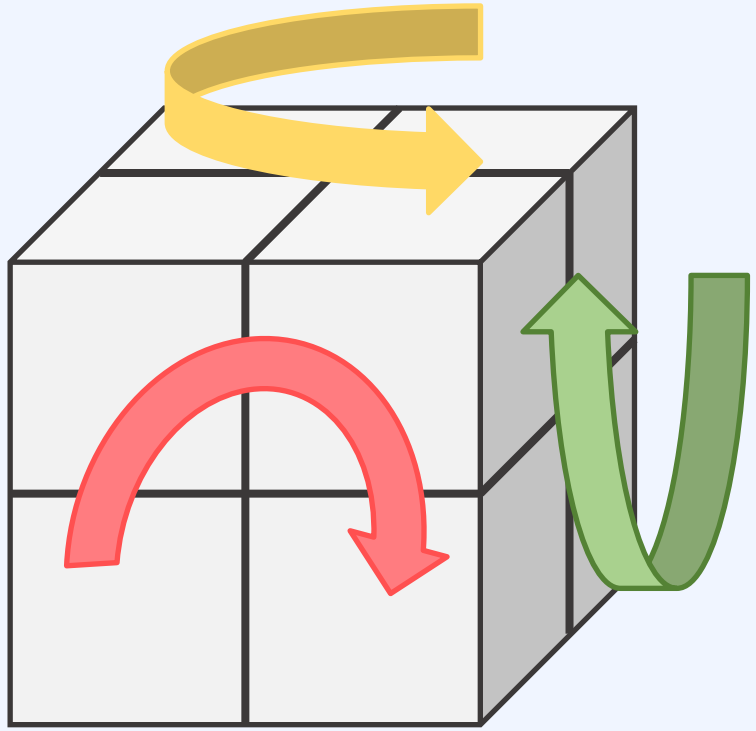
BOJ16939: 2x2x2 큐브

문제 분석

- 큐브는 3가지 축이 있다
  - x축 y축 z축
- 각 축에서 1개의 회전만 구현하면 된다
  - 왜?

BOJ16939: 2x2x2 큐브

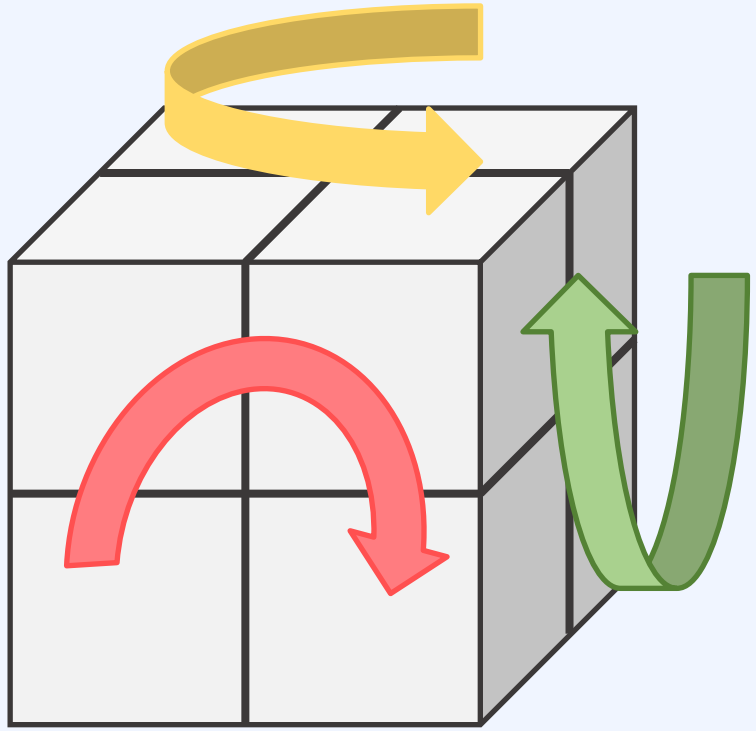
문제 분석



- 시계방향 회전이 가능하다면?
  - 반시계 방향 1회 회전은 시계방향 회전 3회와 같다

BOJ16939: 2x2x2 큐브

문제 분석

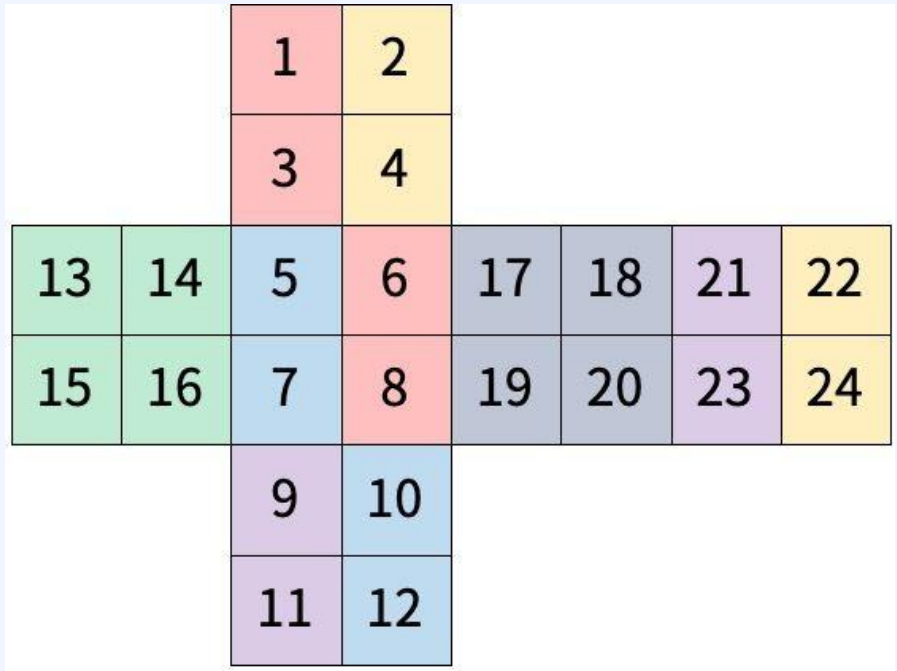


- 큐브에 총 24개 면의 상태가 있다
- 1회 회전 시 각 면이 어떻게 변하는지 기록 (클립 1의 주사위와 유사)

BOJ16939: 2x2x2 큐브

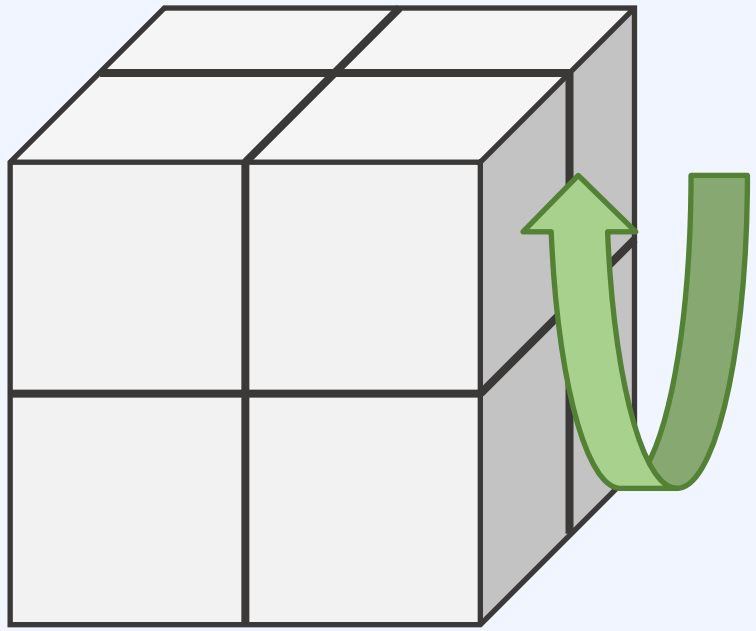
문제 분석

- 각 면에 적힌 수는 문제 에 그려진 전개도를 참조



BOJ16939: 2x2x2 큐브

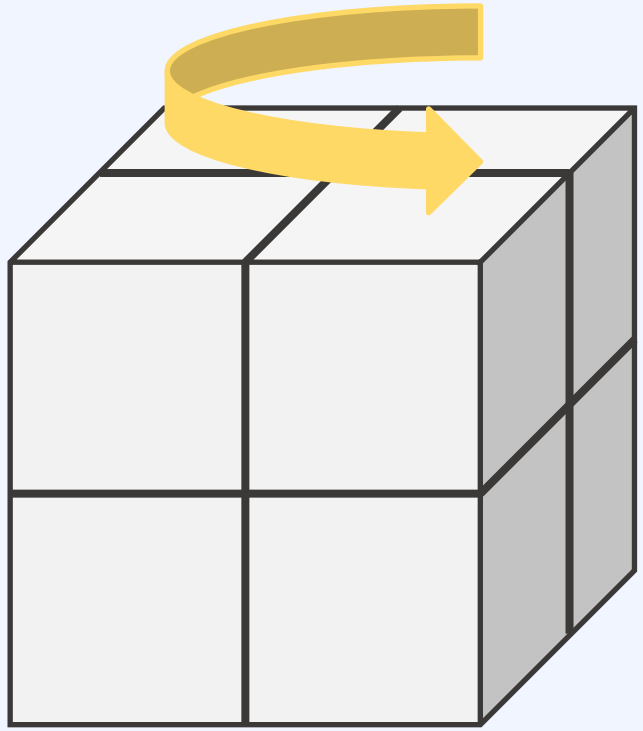
구현



```
void r() {
    int temp1 = area[2];
    int temp2 = area[4];
    area[2] = area[6];
    area[4] = area[8];
    area[6] = area[10];
    area[8] = area[12];
    area[10] = area[23];
    area[12] = area[21];
    area[23] = temp1;
    area[21] = temp2;
}
```

BOJ16939: 2x2x2 큐브

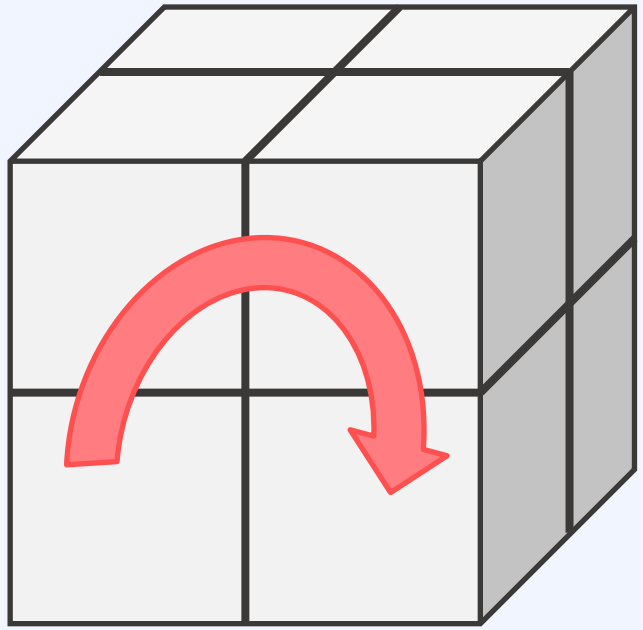
구현



```
void u() {
    int temp1 = area[5];
    int temp2 = area[6];
    area[5] = area[13];
    area[6] = area[14];
    area[13] = area[21];
    area[14] = area[22];
    area[21] = area[17];
    area[22] = area[18];
    area[17] = temp1;
    area[18] = temp2;
}
```

BOJ16939: 2x2x2 큐브

구현



```
void f() {
    int temp1 = area[3];
    int temp2 = area[4];
    area[3] = area[16];
    area[4] = area[14];
    area[16] = area[10];
    area[14] = area[9];
    area[10] = area[17];
    area[9] = area[19];
    area[17] = temp1;
    area[19] = temp2;
}
```

Ch05. 분석과 구현

4. [14599] 인공지능 테트리스 (Large)

BOJ14599: 인공지능 테트리스(Large)

문제 요약

- 20행 10열의 테트리스 판
- 가장 많은 줄을 제거할 수 있는 모양을 넣을 때 제거 가능한 줄을 출력
- 첫 4행은 비어 있음이 보장됨

BOJ14599: 인공지능 테트리스(Large)

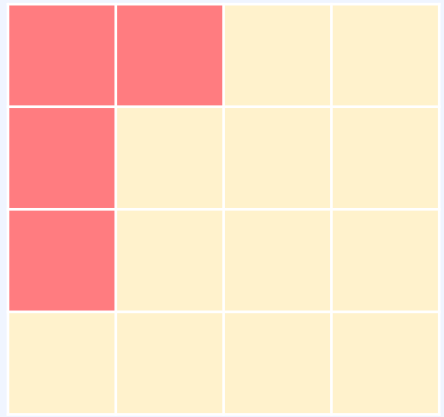
문제 분석

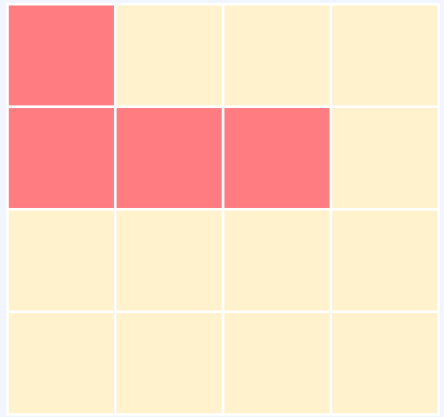
- 구현할 요소가 많은 문제
- 테트리스 블록의 종류는 7가지지만
  회전이 가능하여 총 19종류의 패턴이 생김

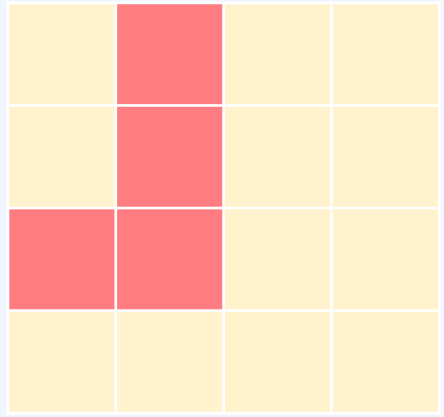
  모든 패턴을 미리 하드 코딩하여 배열에 기록

BOJ14599: 인공지능 테트리스(Large)

문제 분석







```
int[][][] block = {
    {
        {1, 1, 0, 0},
        {1, 0, 0, 0},
        {1, 0, 0, 0},
        {0, 0, 0, 0}
    },
    {
        {1, 0, 0, 0},
        {1, 1, 1, 0},
        {0, 0, 0, 0},
        {0, 0, 0, 0}
    },
    {
        {0, 1, 0, 0},
        {0, 1, 0, 0},
        {1, 1, 0, 0},
        {0, 0, 0, 0}
    },
```

BOJ14599: 인공지능 테트리스(Large)

문제 분석

- 현재 블록을 {하, 좌, 우} 로 이동시키며 탐색
  - 왼쪽으로는 왜 이동해야 할까?
  - 우측 케이스의 정답은 4이다
    - 오른쪽으로 갔다가 다시 왼쪽으로 이동

```
..................
0000000000
0000000000
0000000000
0000000000
1111111110
0000000000
0000000000
0000000000
0000000000
0111111111
0111111111
0111111111
0111111111
```

BOJ14599: 인공지능 테트리스(Large)

문제 분석

- 블록을 중간에 제거하면 안된다
- 우측 케이스의 정답은 2이다
  - 상단의 블록을 4줄을 | 모양으로 제거하면 안된다

```
..................
0111111111
0111111111
0111111111
0111111111
0000000000
0000000000
0000000000
0000000000
0000000000
0000000000
0111111111
0111111111
```

BOJ14599: 인공지능 테트리스(Large)

구현

```
for (int r = 1; r <= 20; r++) {
    String s = sc.next();
    for (int c = 1; c <= 10; c++) {
        arr[r][c] = s.charAt(c - 1) - '0';
    }
}

for (int r = 0; r <= 20; r++) {
    arr[r][0] = 1;
    arr[r][11] = 1;
}
for (int c = 0; c <= 11; c++) {
    arr[21][c] = 1;
}
```

입력 처리

테두리 구간에 1을 넣어
비교 연산을 쉽게 처리

```
static void dfs(int r, int c, int blockType) {
    check[blockType][r][c] = 1;
    for (int i = 1; i <= 3; i++) {
        int nr = r + dr[i];
        int nc = c + dc[i];
        if (nr < 1 || nr > 20 || nc < 1 || nc > 10) continue;
        if (check[blockType][nr][nc] == 1) continue;
        if (game.check(nr, nc, blockType)) {
            dfs(nr, nc, blockType);
        } else if (i == DOWN) {
            game.add(r, c, blockType);
            answer = Math.max(answer, game.bomb(r));
            game.remove(r, c, blockType);
        }
    }
}
```

3방향 탐색 (좌/ 우 / 하)

주의: 배치 가능하더라도 배치는 하지 않음

블록이 다음 탐색에 하강할 수 없다면
1. 현재 위치에 블록을 추가하고
2. 제거 가능 라인을 세고
3. 블록을 제거한다

BOJ14599: 인공지능 테트리스(Large)

구현

```
boolean check(int r, int c, int blockType) {
    boolean valid = true;
    int[][] temp = new int[4][4];
    for (int i = 0; i < 4; i++) {
        for (int j = 0; j < 4; j++) {
            if (board[r + i][c + j] + block[blockType][i][j] > 1) {
                valid = false;
                break;
            }
        }
    }
    return valid;
}
```

(r, c)를 왼쪽 상단으로 두고
테트리스 블록을 배치할수 있는지 판단
(블록, 벽에는 1이 있음)

배치 가능하다면 true 리턴