알튜비튜
동적 계획법

오늘은 '다이나믹 프로그래밍'이라고도 불리는 동적 계획법 알고리즘에 대해 배웁니다.
과거에 구한 해를 현재 해를 구할 때 활용하는 알고리즘이죠. 문제에 많이 나오는 굉장히 중요한 알고리즘 중 하나에요.

동적 계획법

- 특정 범위까지의 값을 구하기 위해 이전 범위의 값을 활용하여 효율적으로 값을 얻는 기법
- 이전 범위의 값을 저장(Memoization)함으로써 시간적, 공간적 효율 얻음

도전 문제 1

#알튜비튜

2240번 : 자두나무 - Gold 5

문제

- 매 초마다, 두 개의 나무 중 하나의 나무에서 열매가 떨어짐
- 매 초마다 어느 나무에서 자두가 떨어질지에 대한 정보가 주어졌을 때, 자두가 받을 수 있는 최대 자두의 개수 출력

제한 사항

- 초 T(1≤T≤1,000)
- 움직일 수 있는 횟수 W(1≤W≤30)

자두나무

#알튜비튜

풀이 방법

- 1. 저장할 요소들에 대해 파악 : 매 초마다의 상황, 현재 몇 번 움직였는지
- 2. 2개의 나무가 있음 -> 인덱스로 처리 가능(이동횟수가 홀수이면 무조건 2번 나무 아래, 짝수이면 무조건 1번 나무 아래에 있기 때문)
- 3. 가능한 경우에 대해 생각

점화식을 세워봅시다

#알튜비튜

- i는 초에 대한 것, j는 이동 횟수에 대한 것

```
for (int i = 1; i <= t; i++)
{
    if (li[i] == 1)
        dp[i][0] = dp[i - 1][0] + 1;
    else
        dp[i][0] = dp[i - 1][0];

    for (int j = 1; j <= w; j++)
    {
        if (li[i] == 2 && j % 2 == 1)
            dp[i][j] = max(dp[i - 1][j - 1], dp[i - 1][j]) + 1;
        else if (li[i] == 1 && j % 2 == 0)
            dp[i][j] = max(dp[i - 1][j - 1], dp[i - 1][j]) + 1;
        else
            dp[i][j] = max(dp[i - 1][j - 1], dp[i - 1][j]);
    }
}
```

점화식을 세워봅시다

#알튜비튜

- i초에 자두가 2번 나무에서 떨어지고, 현재 2번 나무 일 때 (이동 횟수가 홀수면 2번 나무)

```
if (li[i] == 2 && j % 2 == 1)
    dp[i][j] = max(dp[i - 1][j - 1], dp[i - 1][j]) + 1;
```

- i초에 자두가 1번 나무에서 떨어지고, 현재 1번 나무 일 때 (이동 횟수가 짝수면 1번 나무)

```
else if (li[i] == 1 && j % 2 == 0)
    dp[i][j] = max(dp[i - 1][j - 1], dp[i - 1][j]) + 1;
```

점화식을 세워봅시다

#알튜비튜

- 현재 서있는 곳에서 자두를 받을 수 없는 경우

```
else
    dp[i][j] = max(dp[i - 1][j - 1], dp[i - 1][j]);
```

도전 문제 2

2098번 : 외판원 순회 - Gold 1

문제

- 외판원이 어느 한 도시에서 출발해 N개의 도시를 모두 거쳐 다시 원래의 도시로 돌아오는 순회
- 외판원의 순회에 필요한 최소 비용 출력

제한 사항

- 도시의 수 N (2<=N<=16)

외판원 순회

풀이 방법 : dp + dfs + 비트마스킹

- 1. dfs를 활용해 탐색 시작
- 2. dfs 안에서 모든 도시를 방문한 경우, 비용을 구함
- 3. 이전에 계산된 루트의 경우, dp 활용
- 4. 방문할 수 있는 도시에 한해서 dfs 진행
- 5. 현재 경로에 대해 최소 비용 갱신 후 dp에 저장

비트마스킹이란?

- 이진수를 사용하는 컴퓨터의 연산 방식을 이용하여, 정수의 이진수 표현을 자료 구조로 쓰는 기법

ex) 총 5개 도시가 있을 때,

1번과 3번 도시를 방문했다면 = 00101
2번과 4번 도시를 방문했다면 = 01010
모든 도시를 방문하지 않았다면 = 00000

비트마스킹을 사용하는 이유

- 도시를 방문하는 경로의 모든 경우의 수를 찾아야 하기 때문에,
  방문한 도시와 방문하지 않은 도시를 저장 해야함

  → 배열로 저장하면 메모리가 너무 많이 들고, 시간이 오래 걸림

비트마스킹 연산

- 시프트 연산
  - 시프트 연산을 하면 비트를 왼쪽으로 이동시킬 수 있음
  - 왼쪽 시프트 연산을 하면 지정된 위치 수만큼 왼쪽으로 이동
  - ex) 1(3) 이면 1000이 됨 -> (1 << 3)

- OR 연산
  - 각 자리를 계산해서 하나라도 1이면 그 자리는 1이 됨
  - ex) 101 | 010 연산이면 → 111

비트마스킹 연산

- AND 연산
  - 각 자리를 계산해서 모두 1이면 1, 하나라도 0이라면 0이 됨
  - ex) 100 & 011 연산이면 → 000
  - 즉, 왼쪽과 오른쪽 연산에서 겹치는 부분이 하나도 없으면 0이 됨

외판원 순회 비트마스킹 풀이

#알튜비튜

비트마스킹으로 방문한 도시 추가하는 방법

- 도시에 방문했을 때 : 비트 값 1
- 도시에 방문하지 않았을 때 : 비트 값 0

- 방문한 도시 | (1 << 방문한 도시의 번호)

예시 : 5개의 도시가 있고 1번 도시, 3번 도시를 방문 (→ 방문한 도시의 번호로 시프트 연산을 하기 위해 오른쪽에 0 하나 추가)

현재 방문한 도시 비트마스킹 : 001010 → 오른쪽 0은 안쓰는 0이라고 생각하면 됨

2번 도시를 방문

001010 | (1<<2) 즉, 001010 | 100

```
001010
   100
----------
001110
```

외판원 순회 비트마스킹 풀이

#알튜비튜

비트마스킹으로 도시 방문을 확인하기

- 도시에 방문했을 때 : 비트 값 1
- 도시에 방문하지 않았을 때 : 비트 값 0

- 방문한 도시 & (1 << 방문한 도시의 번호)

예시 : 5개의 도시가 있고 1번 도시, 3번 도시를 방문 (→ 방문한 도시의 번호로 시프트 연산을 하기 위해 오른쪽에 0 하나 추가)

현재 방문한 도시 비트마스킹 : 001010 → 오른쪽 0은 안쓰는 0이라고 생각하면 됨

1번 도시를 방문했는지에 대해 검사

001010 | (1<<1) 즉, 001010 | 10

```
001010
000010
----------
000000
```

코드

#알튜비튜

```
int dfs(int x, int visited) // 방문 하지 않은 도시들에 대해 최소 비용을 갱신
{
    if (visited == (1 << n) - 1) // 도시를 전부 방문한 경우
    {
        if (li[x][0]) {//원래 도시로 돌아갈 수 있는 경우
            return li[x][0];
        }
        else {// 원래 도시로 돌아갈 수 없는 경우
            return INF;
        }
    }

    if (dp[x][visited] != -1) {// 이미 계산(방문)된 경우, 그 값을 반환
        return dp[x][visited];
    }

    int min_dist = INF;

    for (int i = 1; i < n; i++)
    {
        if (!(visited & (1 << i)) && li[x][i] != 0) {// 방문 하지 않은 도시의 경우, 최소 비용을 갱신
            min_dist = min(min_dist, dfs(i, visited | (1 << i)) + li[x][i]);
        }
    }

    dp[x][visited] = min_dist;
    return min_dist;
}
```

/<> 20923번 : 숫자 할리갈리 게임 – Silver 1

문제

1. 게임 시작 시 도도와 수연이는 N장의 카드로 구성된 덱을 받음 (그라운드는 비어 있음)
2. 도도와 수연이는 차례대로 덱의 가장 위에 있는 카드를 그라운드에 내려놓음
3. 종을 치는 사람이 그라운드의 카드 더미를 모두 가져감
   a. 수연: 그라운드 위의 카드의 숫자 합이 5가 될 때 종을 침
   b. 도도: 카드의 숫자가 5가 나올 때 종을 침
4. 카드 더미를 가져갈 때는 상대방의 그라운드, 자신의 그라운드 순으로 가져와 자신의 덱 아래에 합침
5. M번 게임을 진행한 후 더 많은 카드를 가진 사람이 승리, 카드의 수가 같은 경우 비김
   a. 게임 도중 덱에 있는 카드가 다 떨어지면 상대가 승리
6. 2~4 과정을 진행하는 것을 한 번 진행한 것으로 볼 때 M번 진행 후 승리한 사람은?

/<> 20923번 : 숫자 할리갈리 게임 - Silver 1

제한 사항

- 도도와 수연이가 가지는 카드의 개수의 범위는 1 <= N <= 30,000
- 게임 진행 횟수의 범위는 1 <= M <= 2,500,000
- 각각의 카드는 1이상 5이하의 자연수가 적혀 있음

예제

#알튜비튜

예제 입력1

```
10 12
1 2
2 2
1 2
2 3
3 1
2 2
2 5
2 1
5 1
2 3
```

예제 출력1

```
do
```

예제 입력2

```
1 1
5 2
```

예제 출력2

```
su
```

예제 입력3

```
3 4
1 2
2 2
1 1
```

예제 출력3

```
dosu
```

순서대로 게임을 진행해보자

M = 0

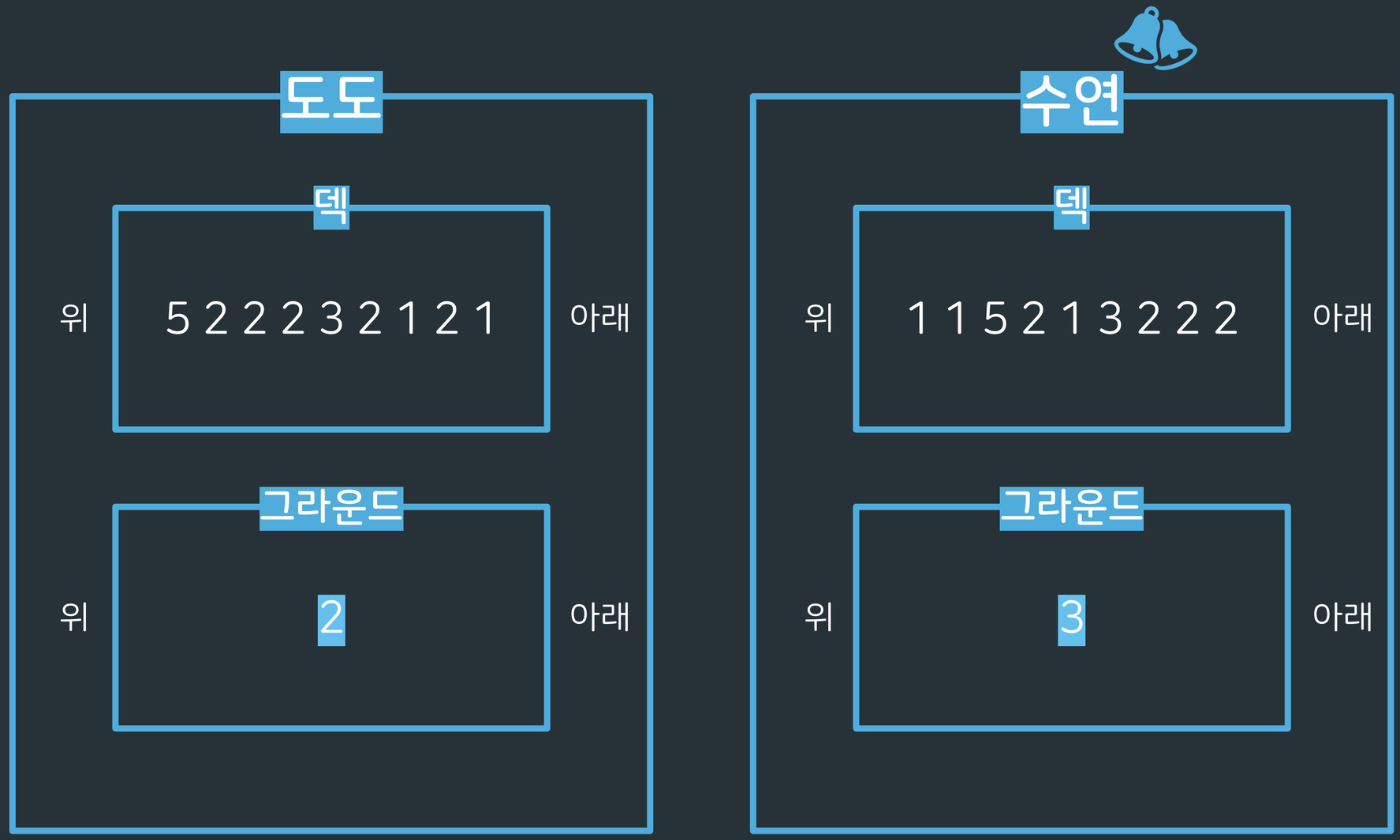
순서대로 게임을 진행해보자

M = 1

순서대로 게임을 진행해보자

#알튜비튜

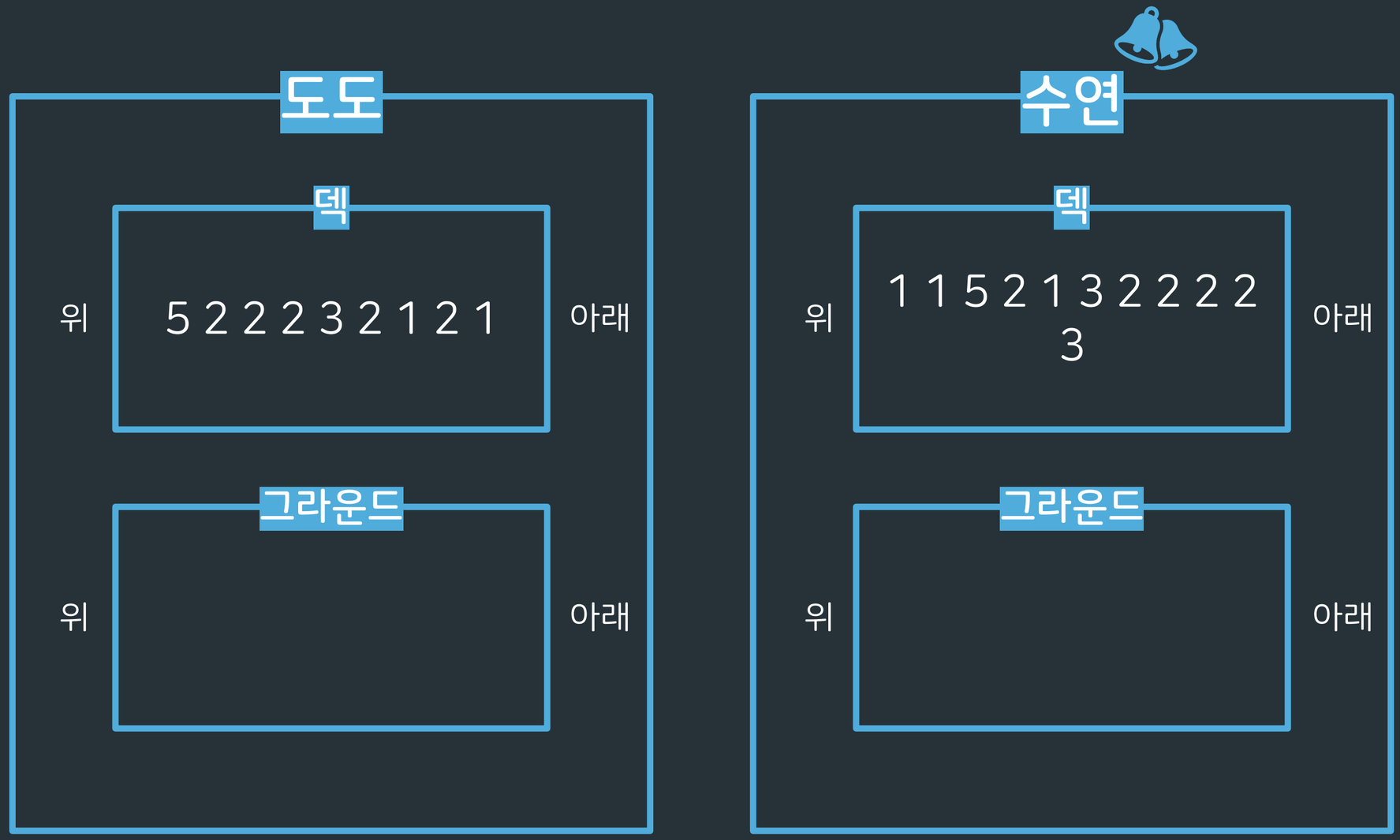
M = 2



순서대로 게임을 진행해보자

#알튜비튜

M = 2

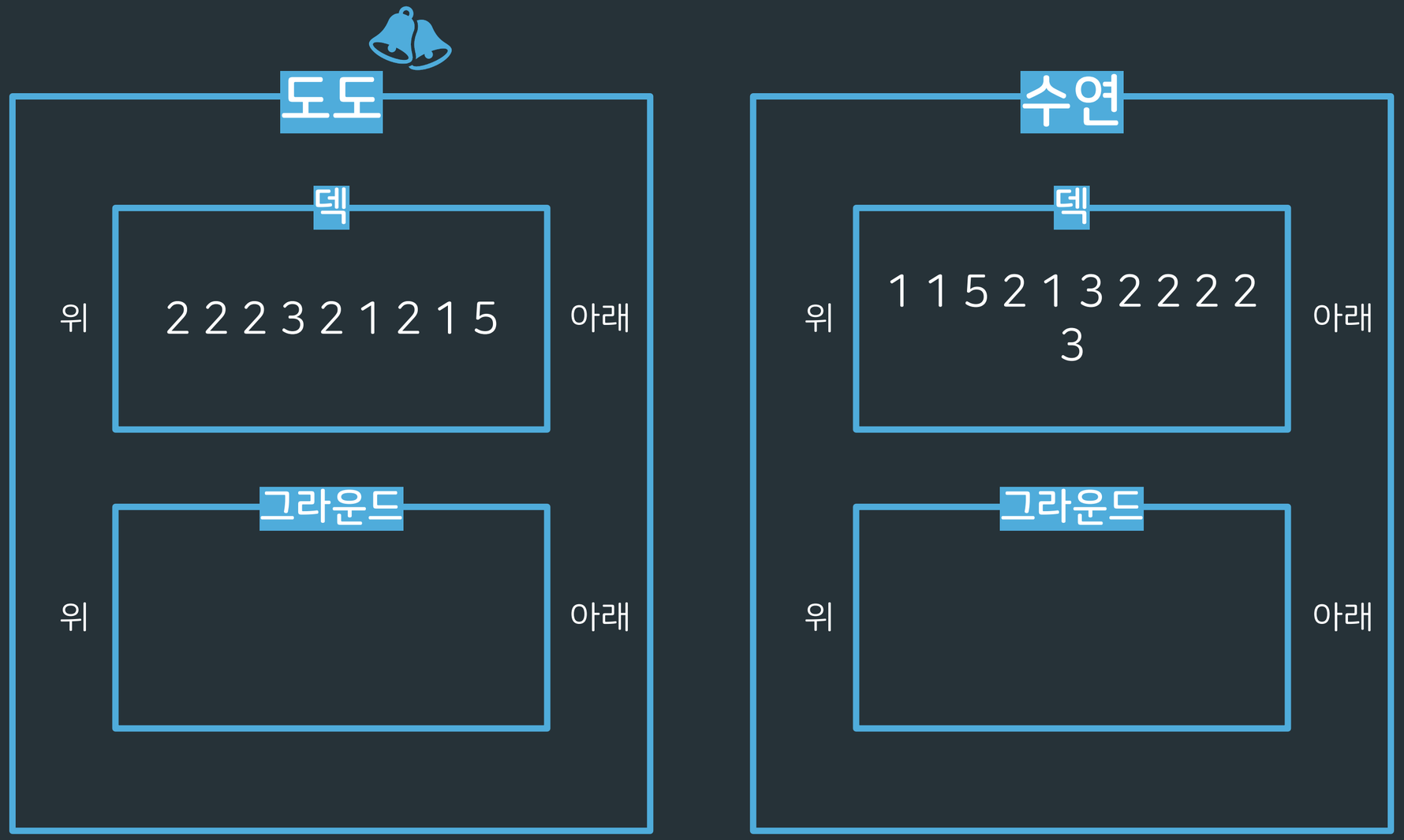


순서대로 게임을 진행해보자

M = 3

순서대로 게임을 진행해보자

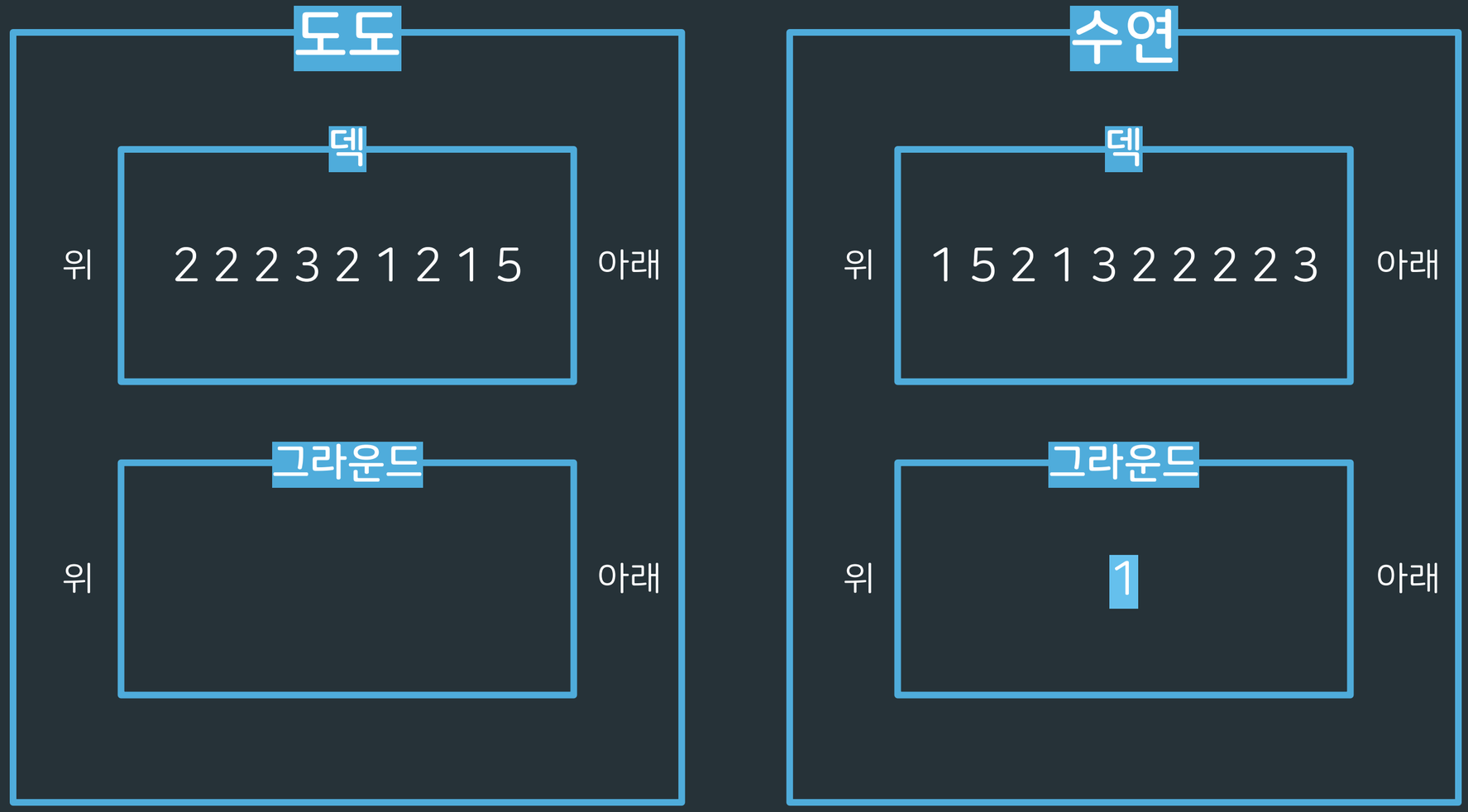
M = 3



순서대로 게임을 진행해보자

#알튜비튜

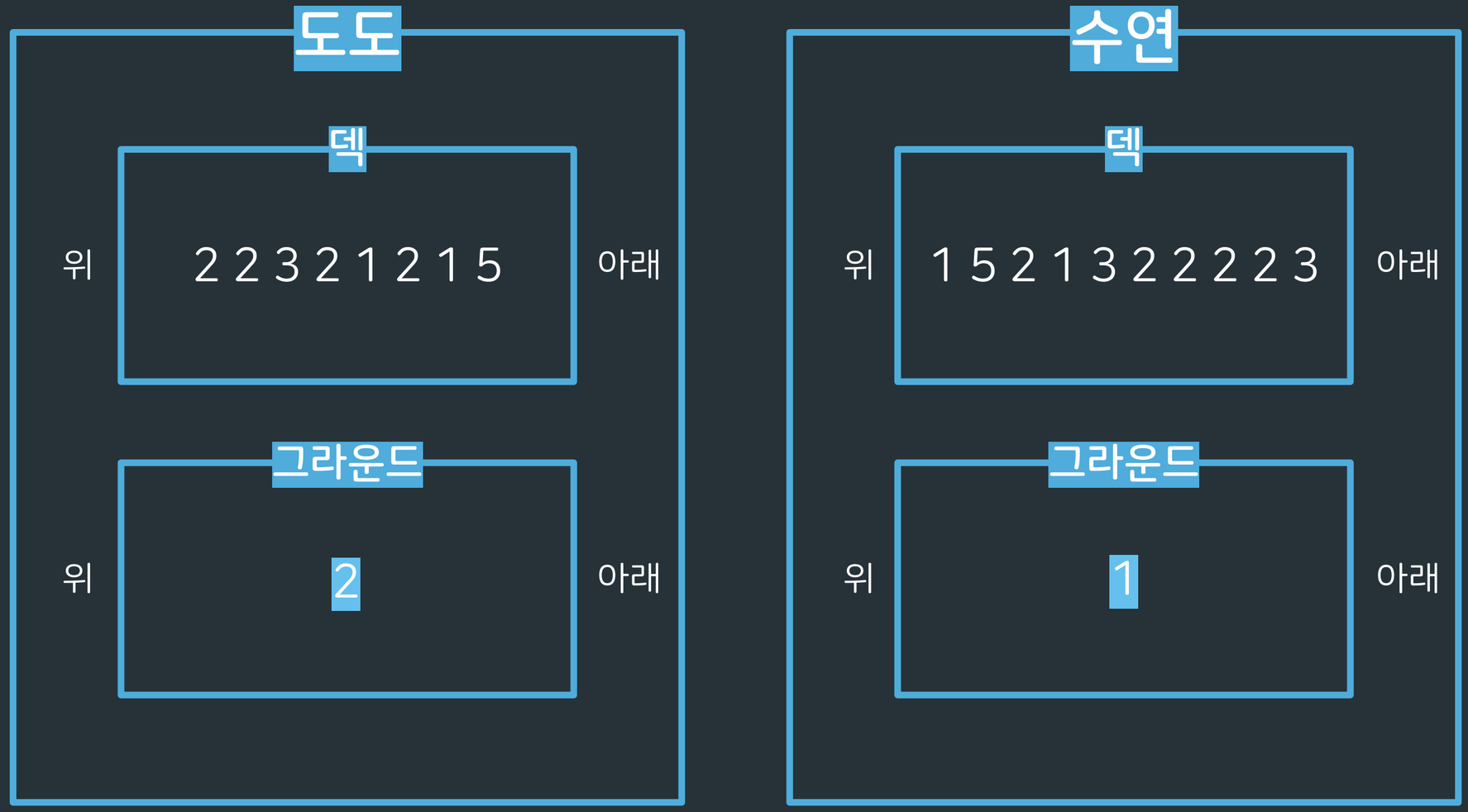
M = 4



순서대로 게임을 진행해보자

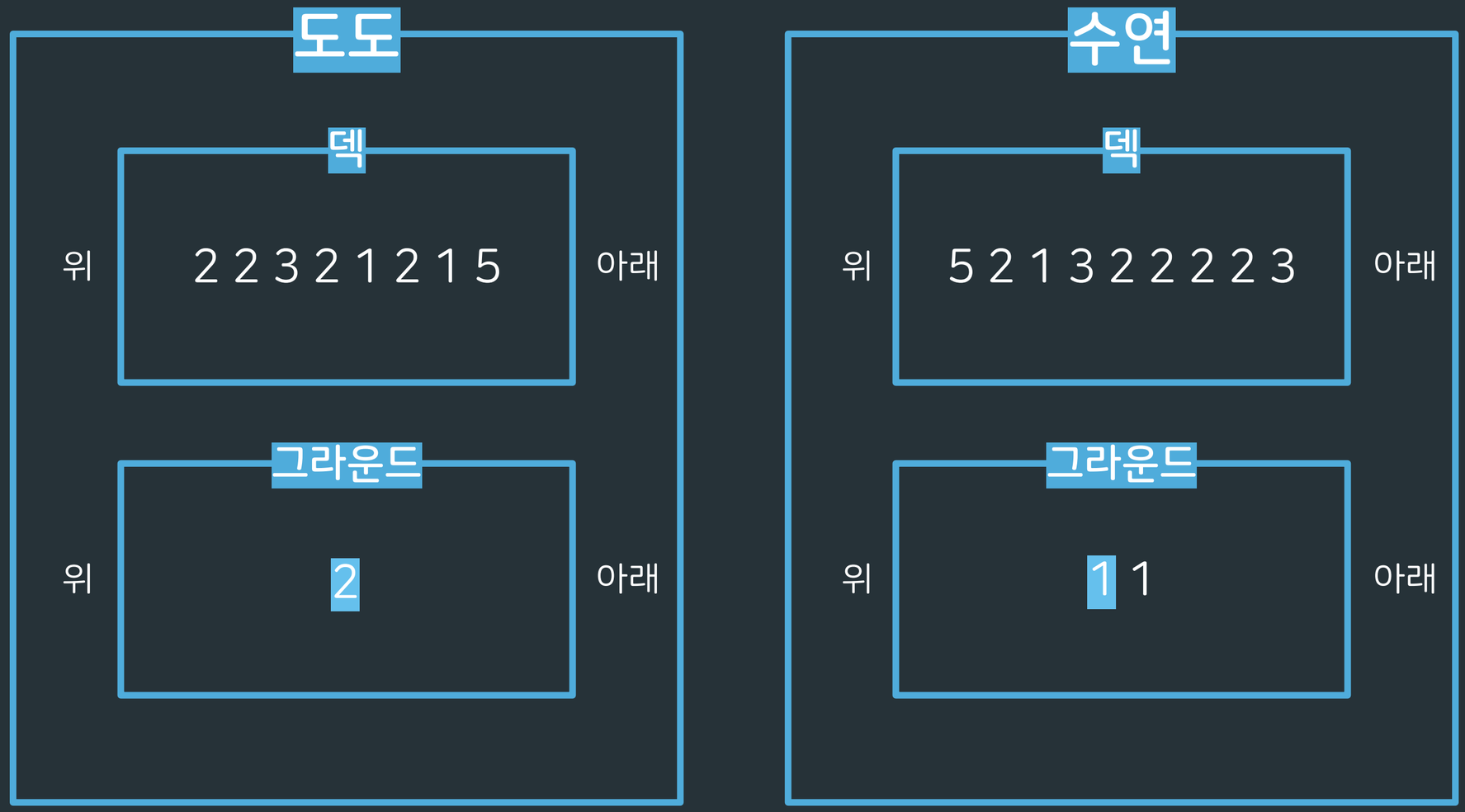
#알튜비튜

M = 5



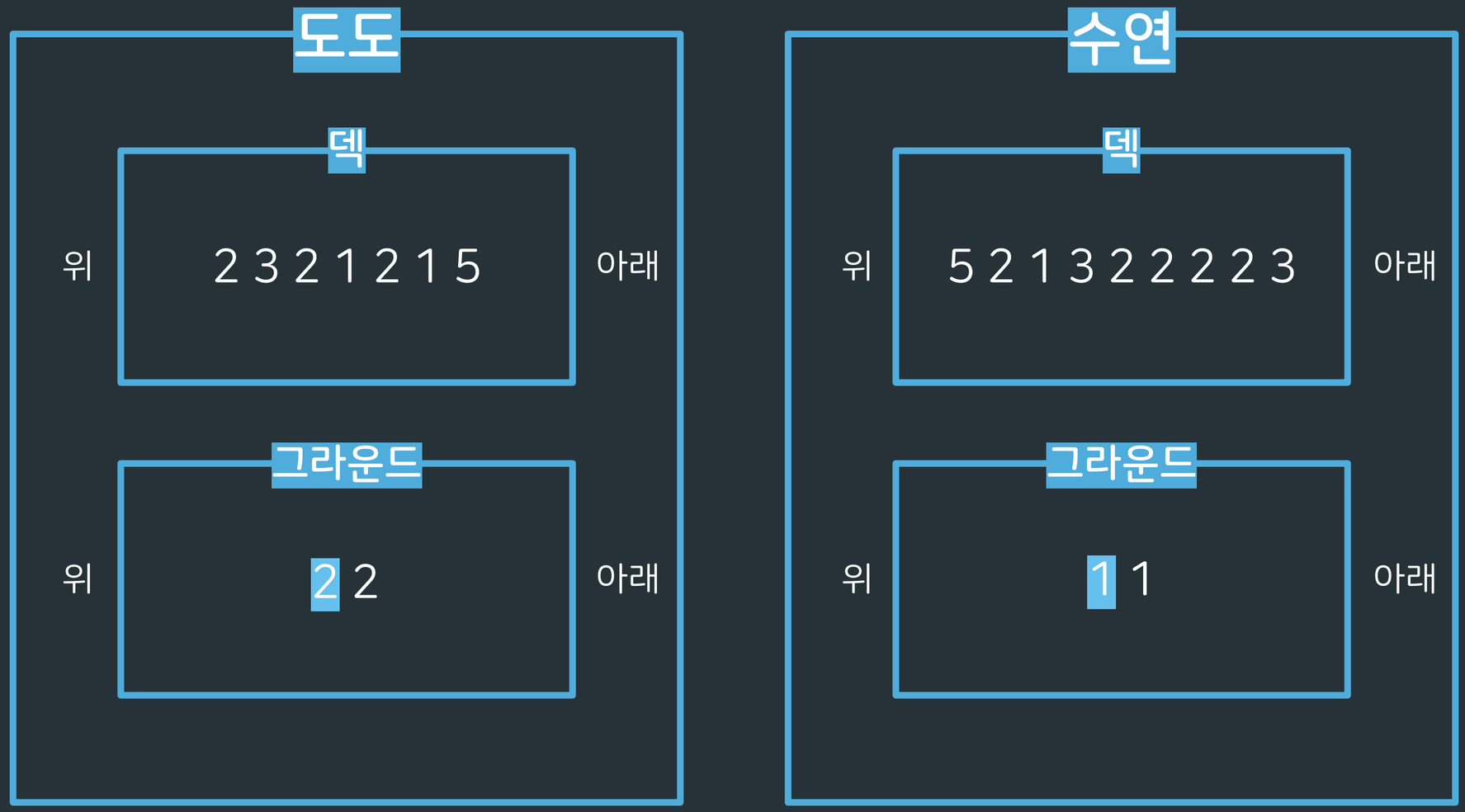
순서대로 게임을 진행해보자

M = 6



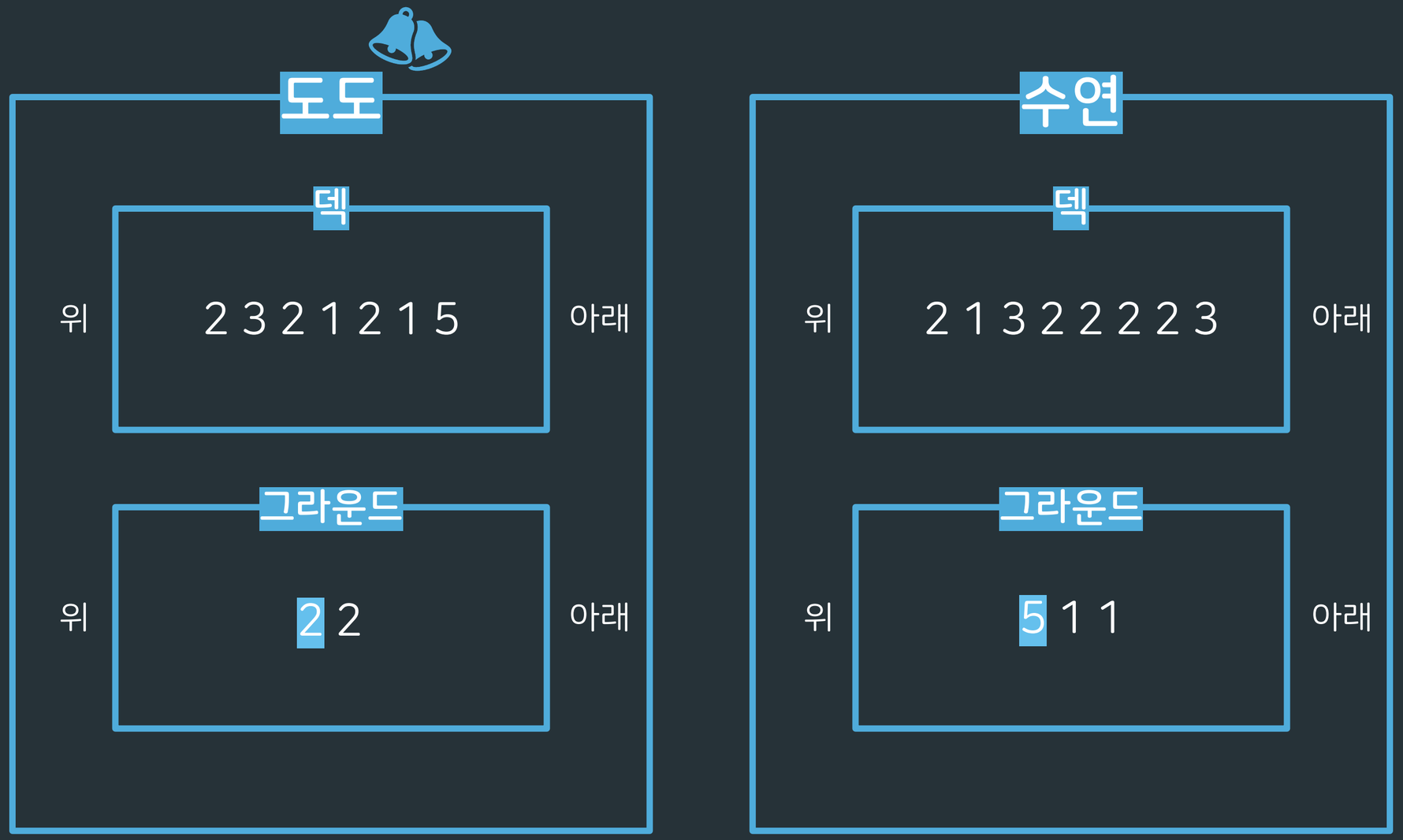
순서대로 게임을 진행해보자

M = 7



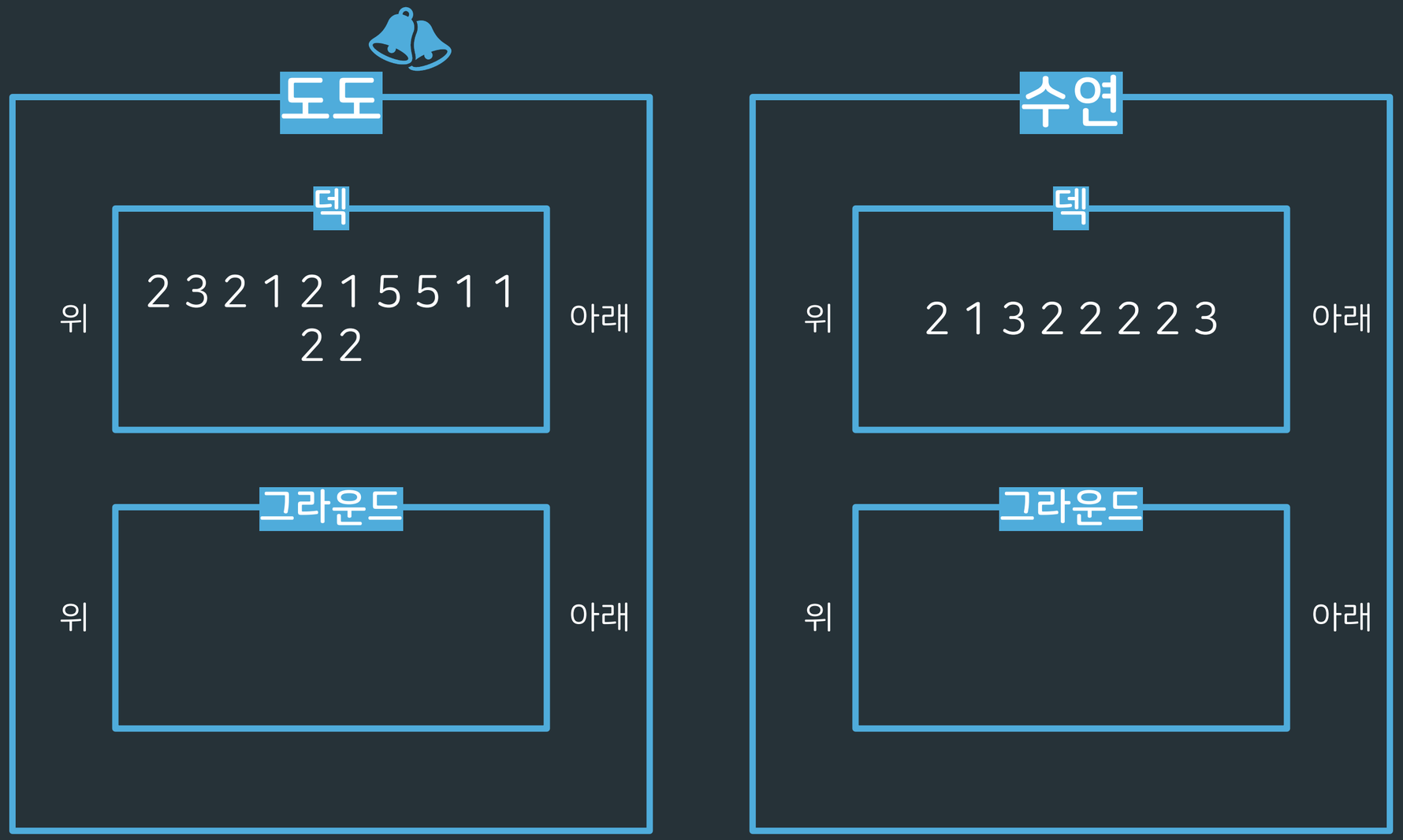
순서대로 게임을 진행해보자

M = 8



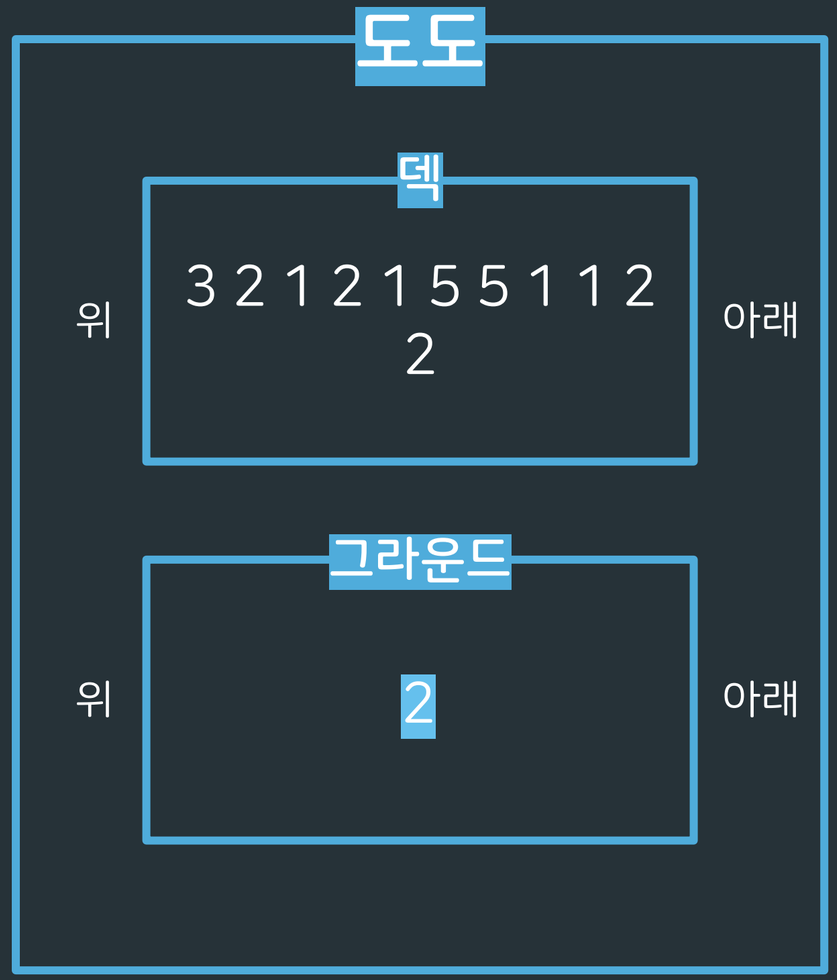
순서대로 게임을 진행해보자

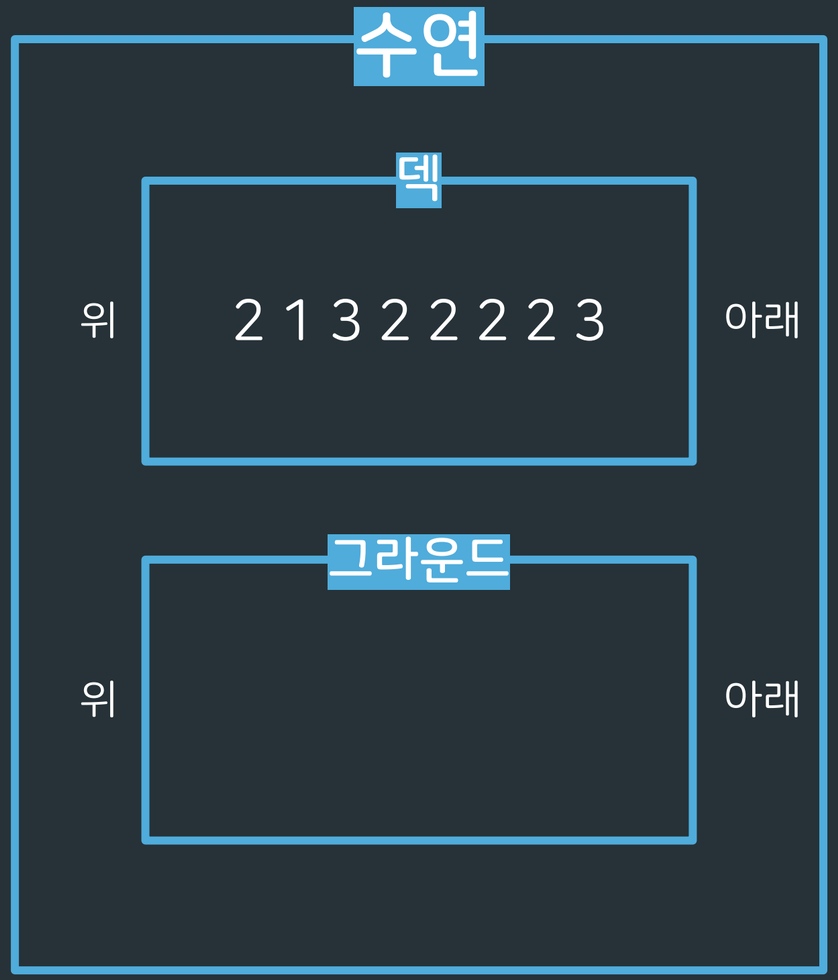
M = 8



순서대로 게임을 진행해보자

M = 9





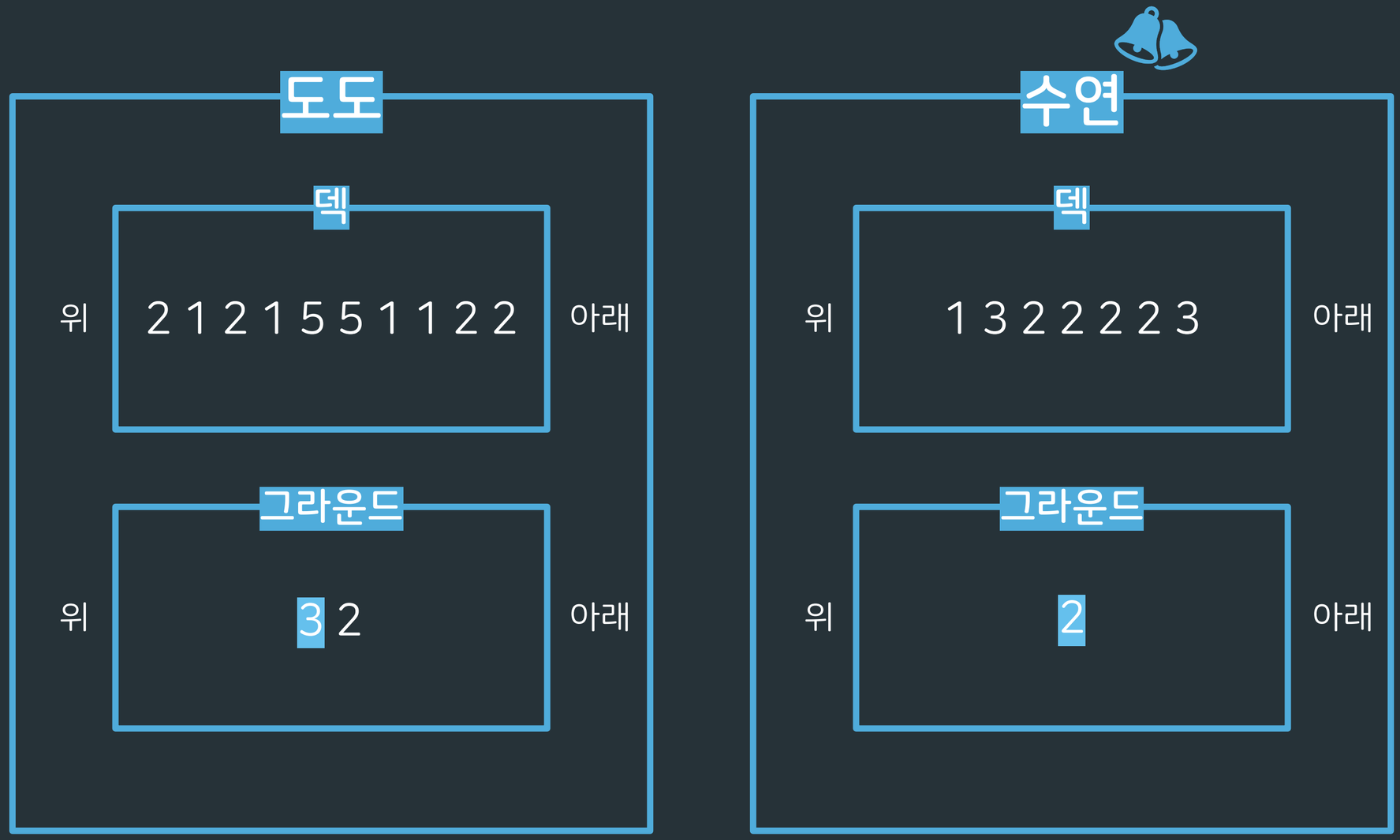
순서대로 게임을 진행해보자

M = 10

순서대로 게임을 진행해보자

#알튜비튜

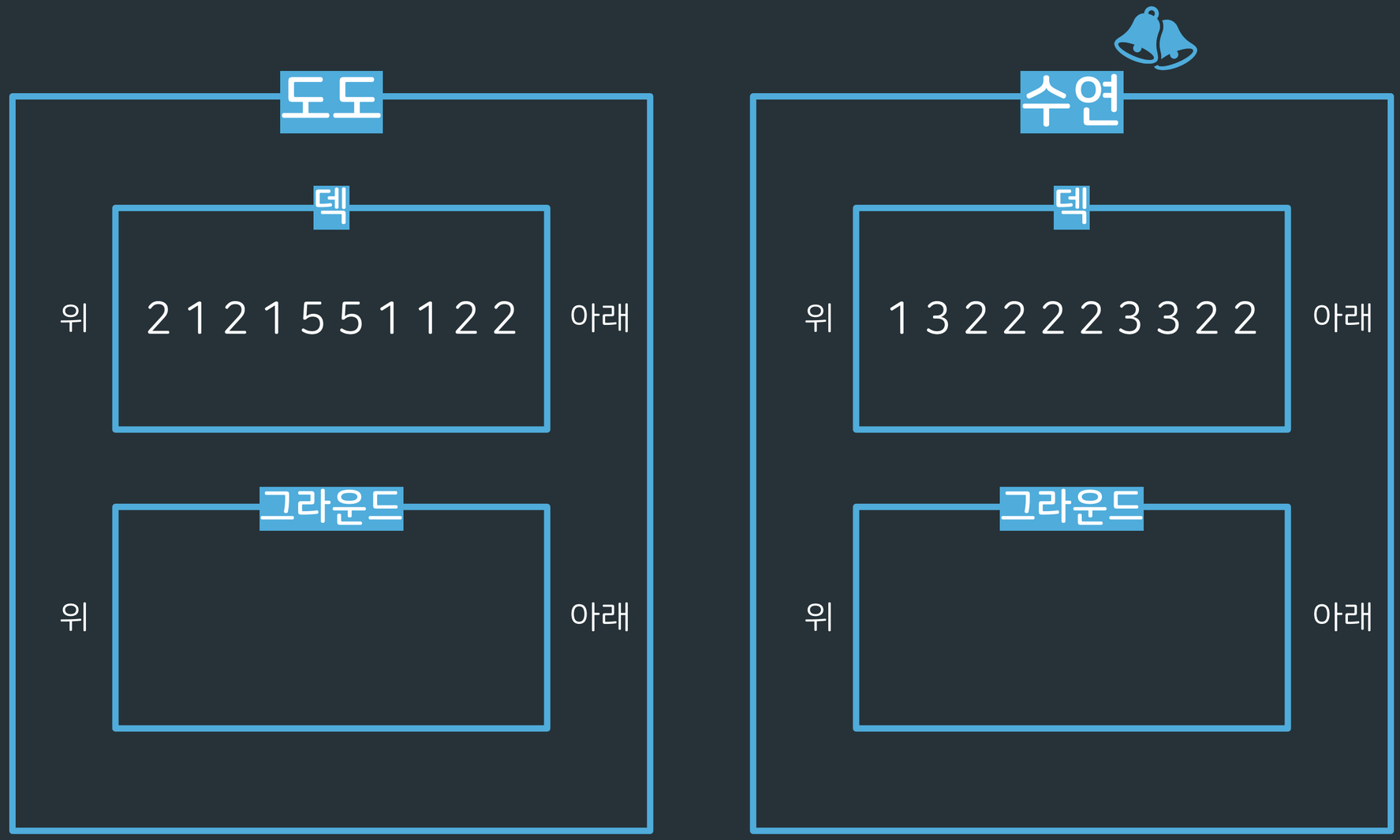
M = 11



순서대로 게임을 진행해보자

#알튜비튜

M = 11



순서대로 게임을 진행해보자

M = 12

순서대로 게임을 진행해보자

10장 > 9장

카드를 더 많이 가지고 있는 도도의 승리!

코드

#알튜비튜

도도, 수연은 각각 덱과 그라운드를 가짐(deque)

1. 카드를 덱에서 한 장씩 내려 놓음 (도도 먼저)
2. 어떤 플레이어가 종을 칠 수 있는지 판단 (도도/수연/없음)
    a. 종을 친 경우 그라운드의 카드를 덱으로 이동
3. 종료 조건 만족 시 승리한 사람 판단

마무리

#알튜비튜

추가로 풀어보면 좋은 문제!

- 9655번 : 돌 게임 – Silver 5
- 9095번 : 1, 2, 3 더하기 – Silver 3
- 2156번 : 포도주 시식 – Silver 1
- 9251번 : LCS – Gold 5