

# 알튜비튜 백트래킹

오늘은 코딩테스트에 많이 나오는 알고리즘 중 하나인 백트래킹에 대해 배웁니다.  
지난 시간에 배운 완전탐색에서 조금 더 발전해서, 더 이상 가망 없는 후보를 제외하고  
탐색하는 알고리즘이죠.

## 백트래킹

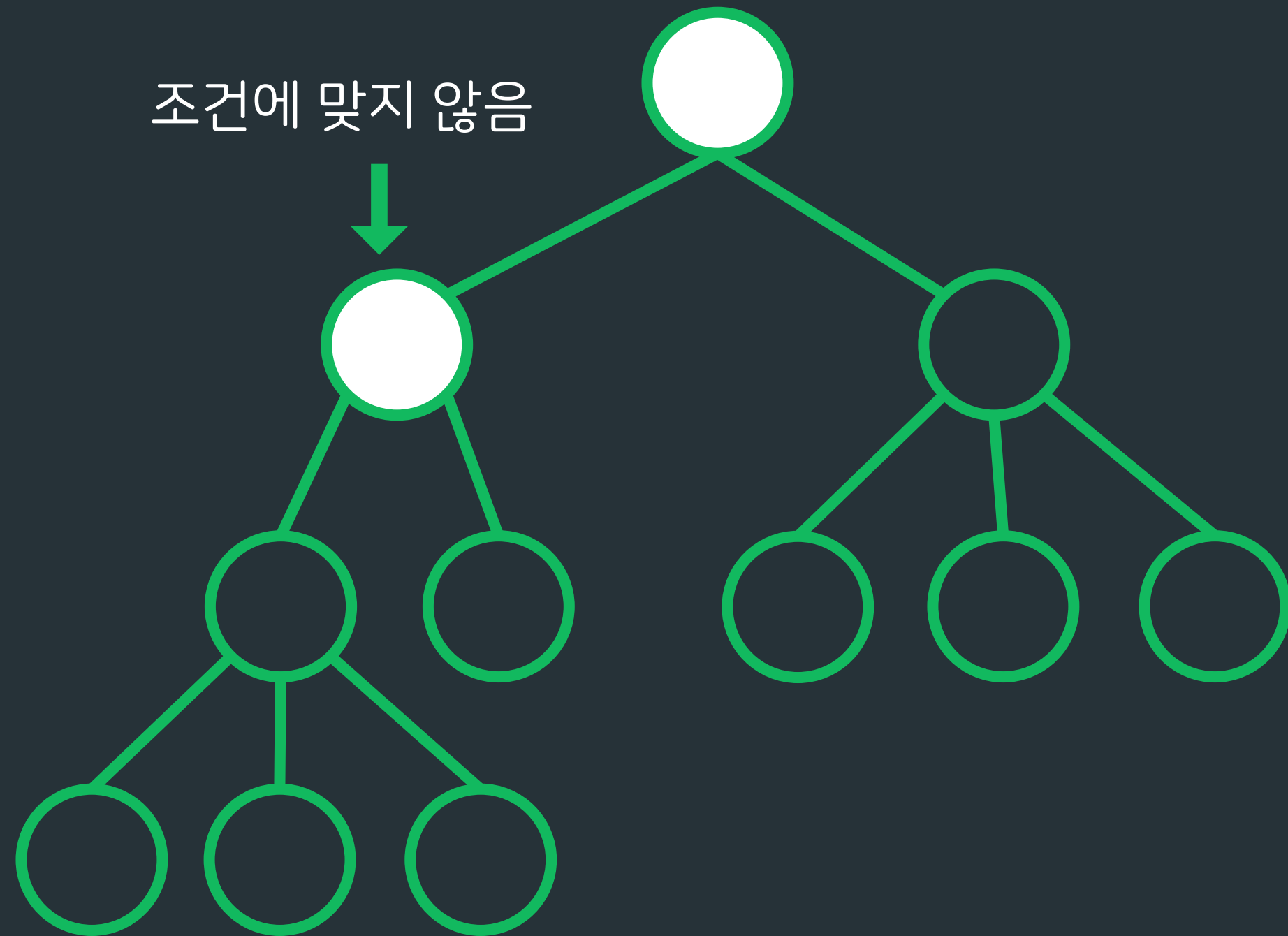
- 완전탐색처럼 모든 경우를 탐색하나, 중간 과정에서 조건에 맞지 않는 케이스를 가지치기하여 탐색 시간을 줄이는 기법
- 모든 경우의 수를 탐색하지 않기 때문에 완전탐색보다 시간적으로 효율적임
- 탐색 중 조건에 맞지 않는 경우 이전 과정으로 돌아가야 하기 때문에, 재귀를 사용하는 경우 많음
- 조건을 어떻게 설정하고, 틀렸을 시 어떤 시점으로 돌아가야 할지 설계를 잘 하는 것이 중요

## 과정

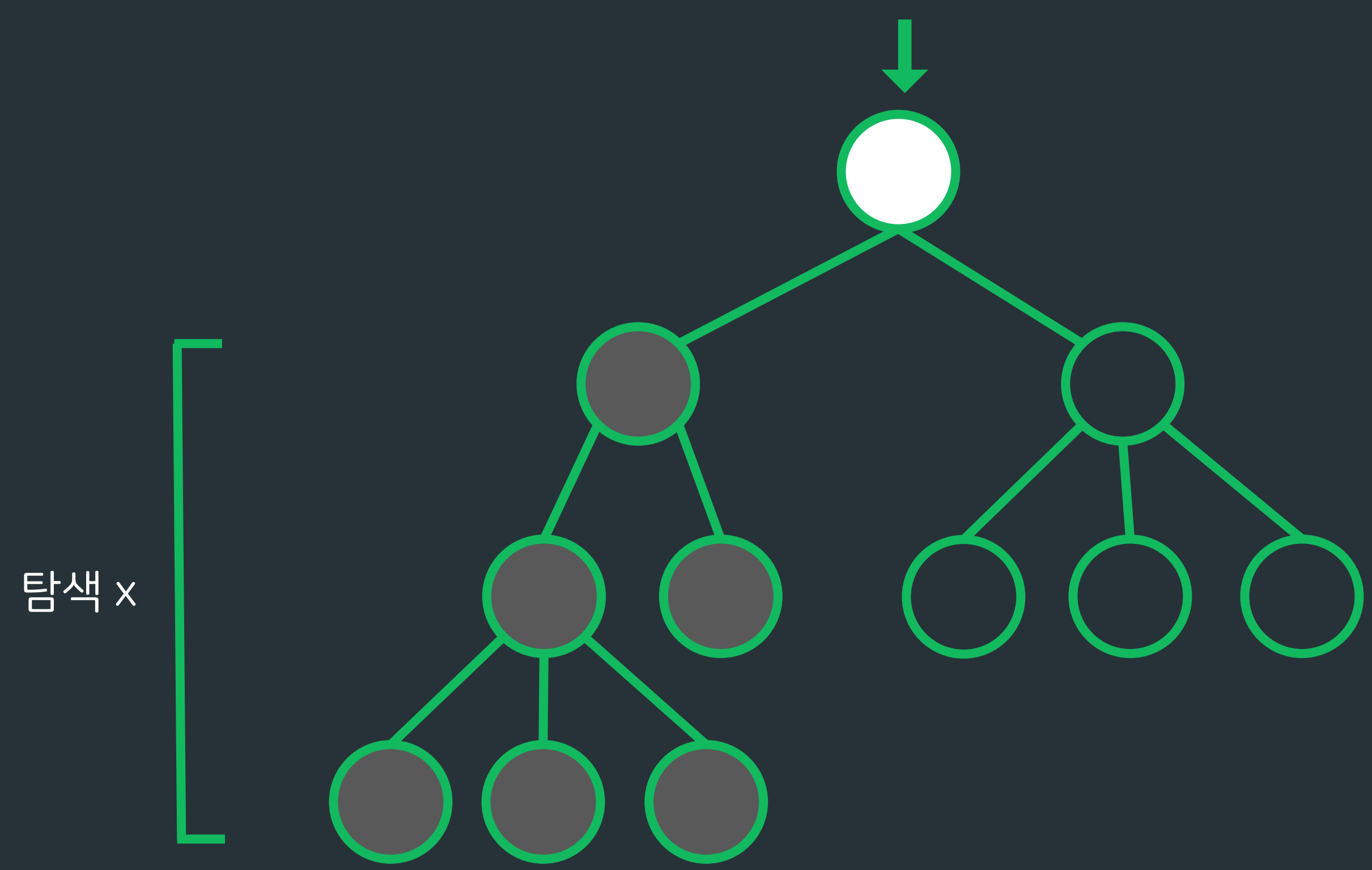
- 어떤 노드의 유망성(promising)을 점검 → 조건에 맞는지 안맞는지
- 유망하지 않다면(non-promising) 배제함 (가지치기)
- 해당 노드의 부모노드로 되돌아간 후 다른 자손노드를 탐색

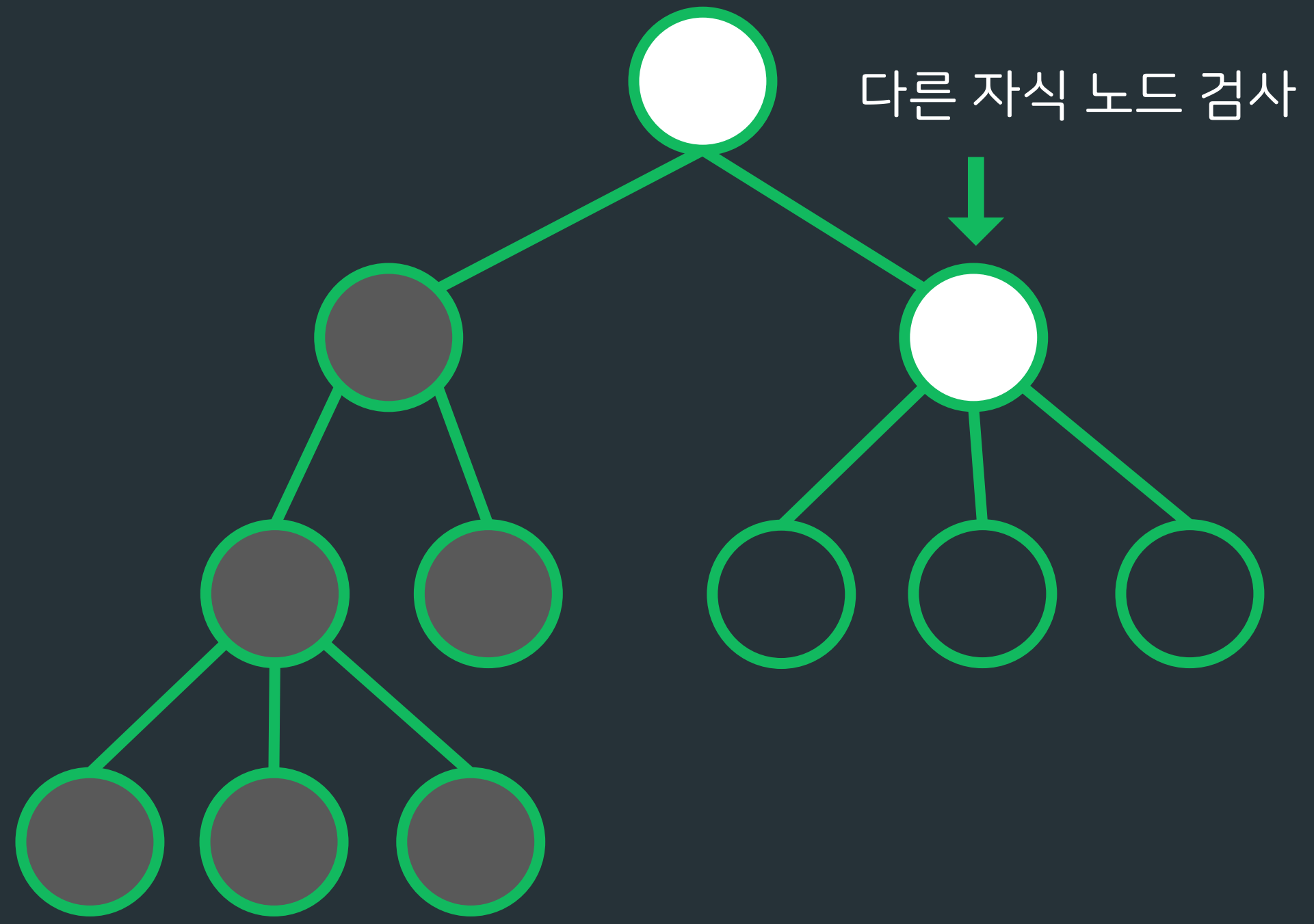
## 가지치기

- 지금의 경로가 해가 될 것 같지 않으면(non-promising)  
그 전으로 되돌아 가는 것(back)
- 즉, 불필요한 부분을 쳐내는 것
- 되돌아간 후 다시 다른 경로 검사
- 가지치기를 얼마나 잘하느냐에 따라 효율성이 결정됨



부모 노드로 돌아감





“즉, **답이 될 만한지(promising)** 판단하고, 그렇지 않다면  
탐색하지 않고 **다시 전으로 넘어가서(back)** 탐색을 계속 하는 것 ”



# 재귀함수란?

```
void f(int mine){  
    f(mine + 1);  
}
```

← 문제점은?

- 자기 자신을 호출하는 함수

# 재귀함수란?

```
void f(int mine){  
    if (mine == 5)  
        return;  
    f(mine + 1);  
    cout << mine << '\n';  
}
```

- 자기 자신을 호출하는 함수
- 가장 중요한 점은 기저 조건(탈출 조건)을 잘 세우는 것!

# 어떨 때 백트래킹을 적용하지?

## 백트래킹

- 주로 재귀함수로 구현하기에 문제에서 주어지는 **N의 크기가 작다!**
- 그 전 과정으로 돌아가면서 하는 **탐색**이 필요한 경우
- **Check 배열**을 많이 사용

## /<> 15649번 : N과 M(1) – Silver 3

### 문제

- 자연수  $n$ ,  $m$ 이 주어짐
- 1부터  $n$ 까지의 자연수 중 중복없이  $m$ 개를 고른 수열을 모두 구하는 문제

### 제한 사항

- 입력 범위는  $1 \leq m \leq n \leq 8$

## 예제 입력

4 2

## 예제 출력

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

## /<> 15649번 : N과 M(1) - Silver 3

### 문제

- 자연수  $n$ ,  $m$ 이 주어짐
- 1부터  $n$ 까지의 자연수 중 중복없이  $m$ 개를 고른 수열을 모두 구하는 문제

### 제한 사항

- 입력 범위는  $1 \leq m \leq n \leq 8$

### 접근

- 제약 조건을 살펴보자  
→ 중복  $x$ , 수열 길이가  $m$
- 재귀함수를 설계해보자  
→ 각 수를 넣을 때, 이미 수열 내에 있으면 넘어감 (체크해줄 무언가가 필요)  
→ 기저조건은 길이가  $m$ 일 때!

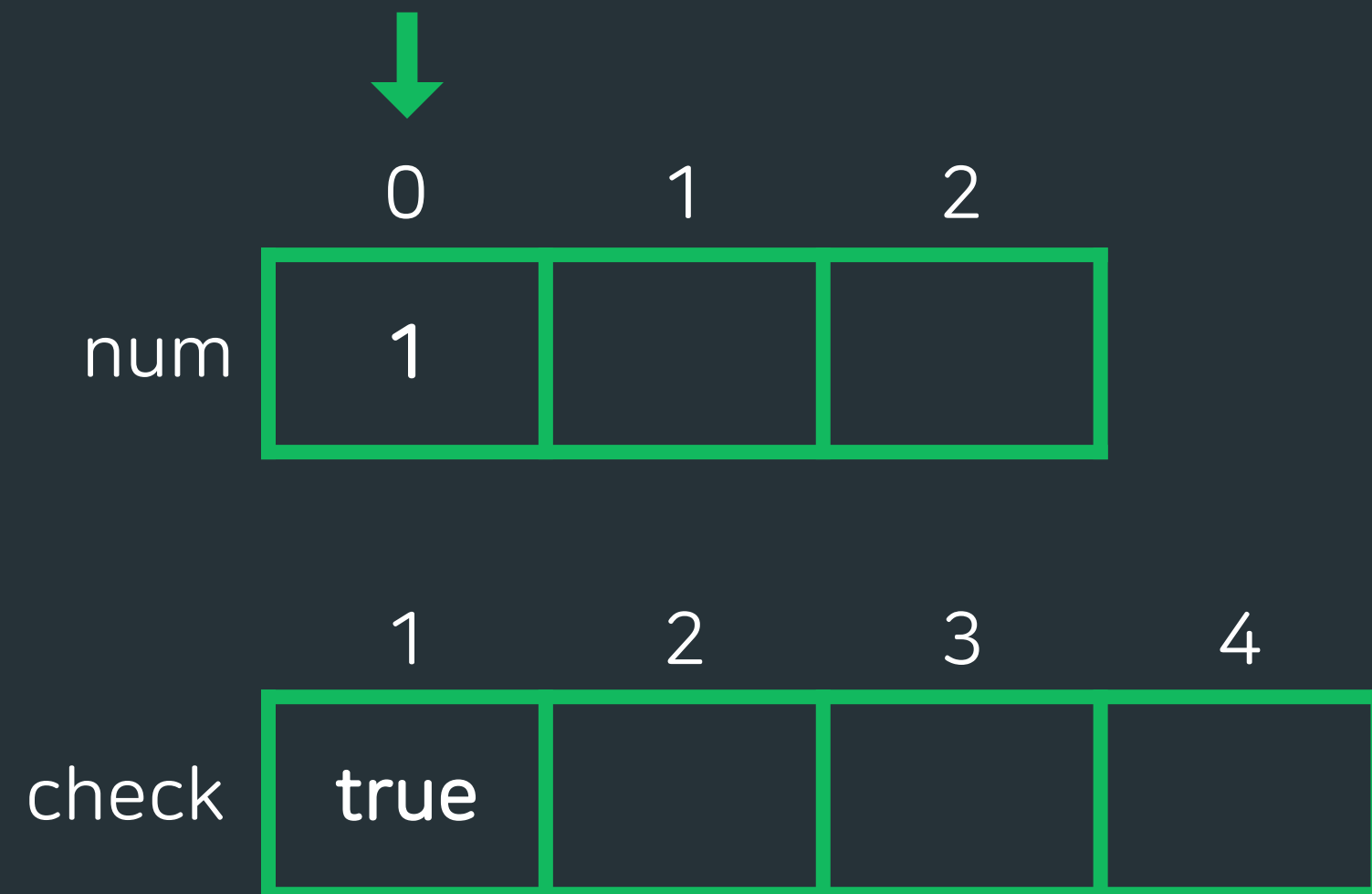
# 체크해줄 무언가?

## 접근

- 해당 수가 현재 수열에서 사용이 되었는지 안되었는지 **체크하는 배열**을 만들자!
- 즉, **수를 인덱스로** 가지는 체크배열
- 이는 곧, **가지치기**의 역할

## N과 M(1)

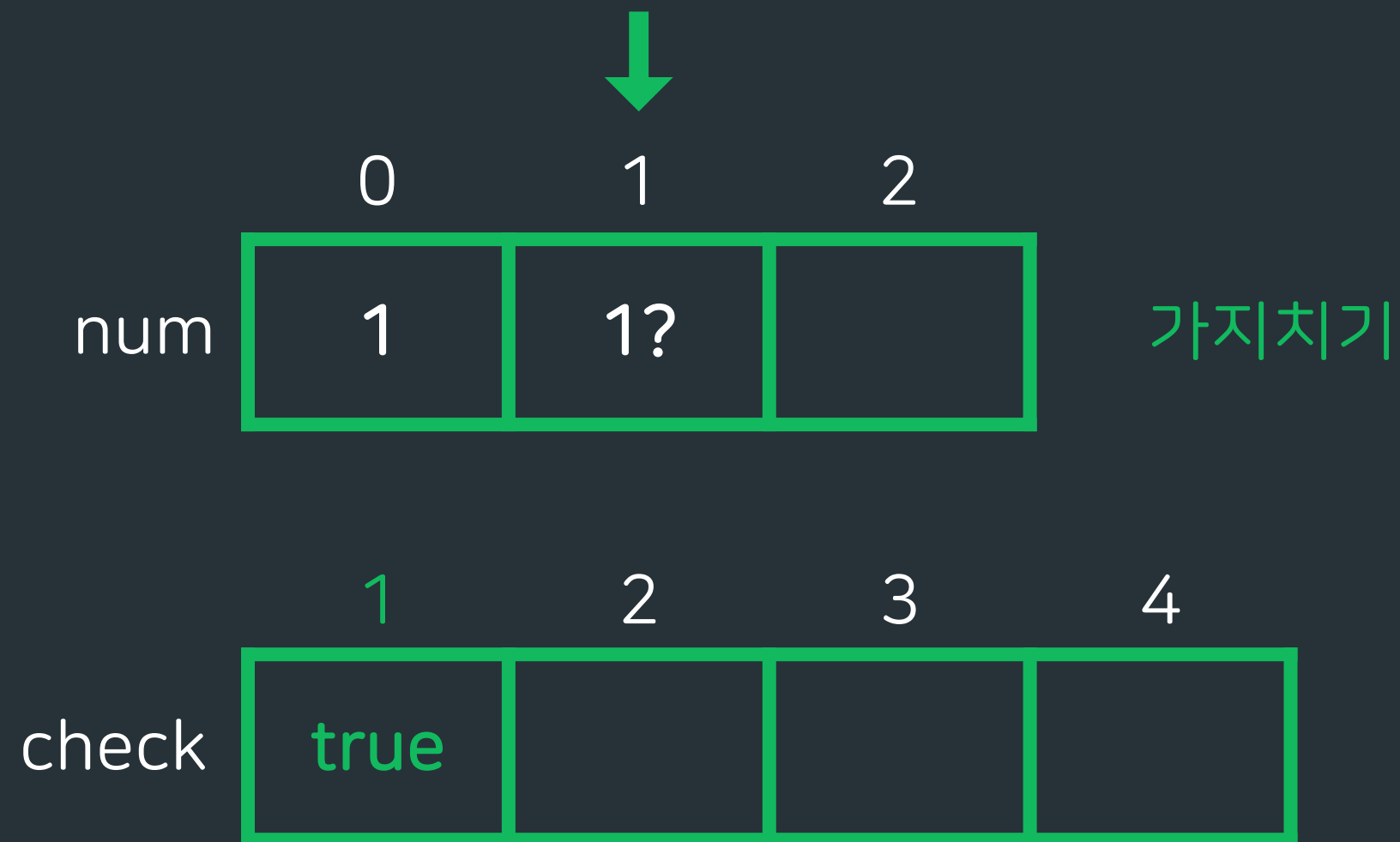
- 1부터  $n$ 까지의 자연수 중 **중복없이**  $m$ 개를 고른 수열을 모두 구하는 문제
- $n = 4, m = 2$





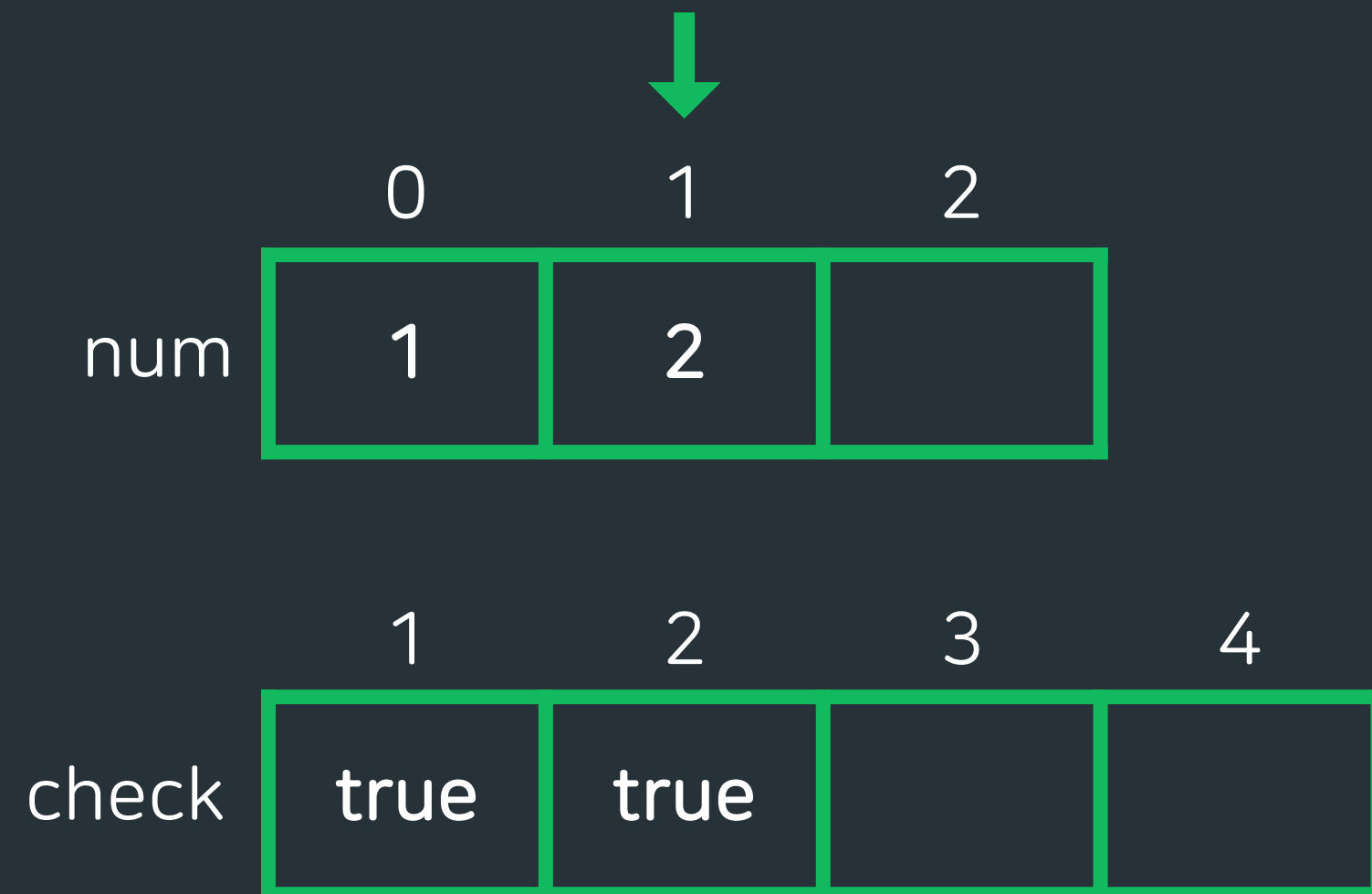
## N과 M(1)

- 1부터  $n$ 까지의 자연수 중 **중복없이**  $m$ 개를 고른 수열을 모두 구하는 문제
- $n = 4, m = 2$



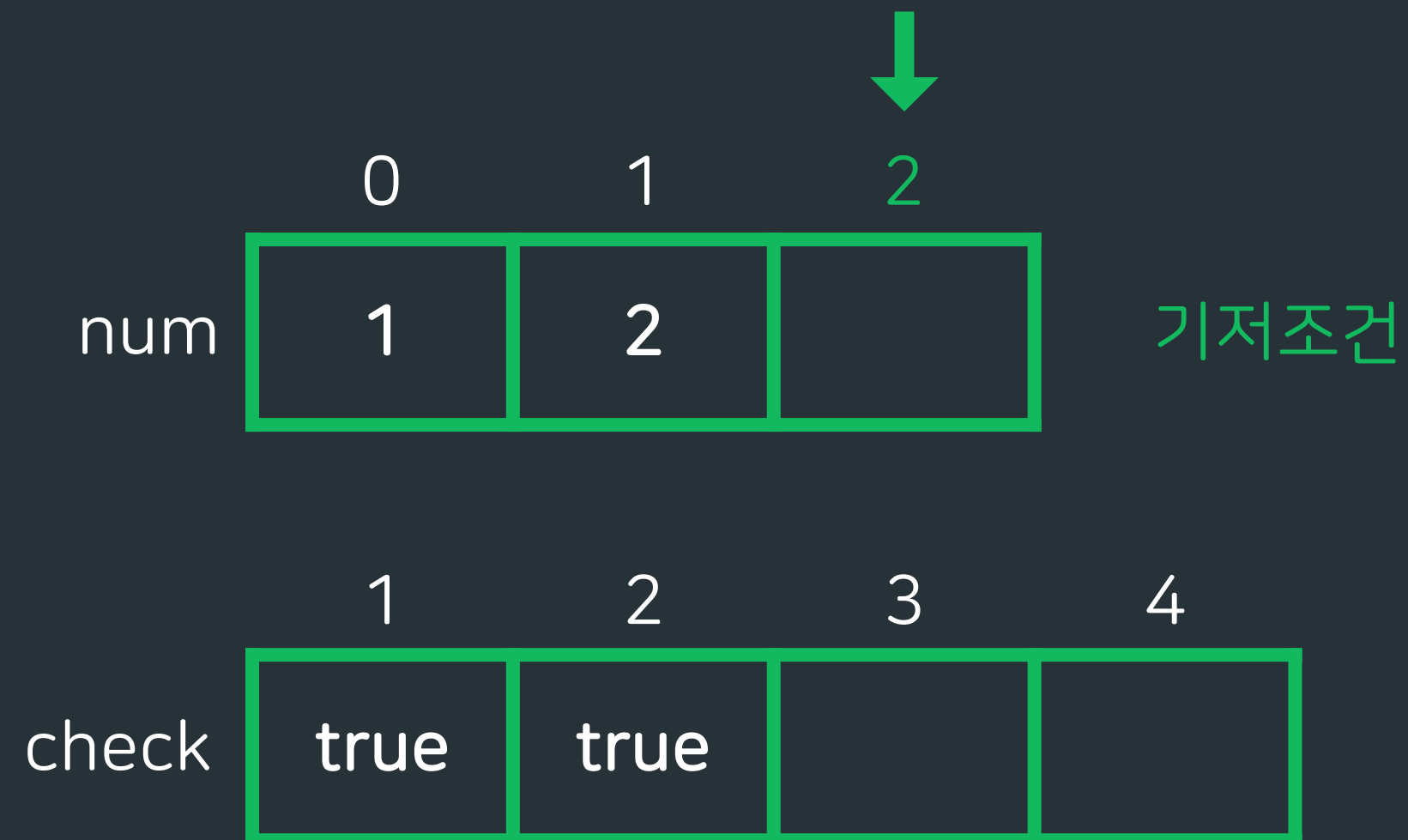
## N과 M(1)

- 1부터  $n$ 까지의 자연수 중 중복없이  $m$ 개를 고른 수열을 모두 구하는 문제
- $n = 4, m = 2$



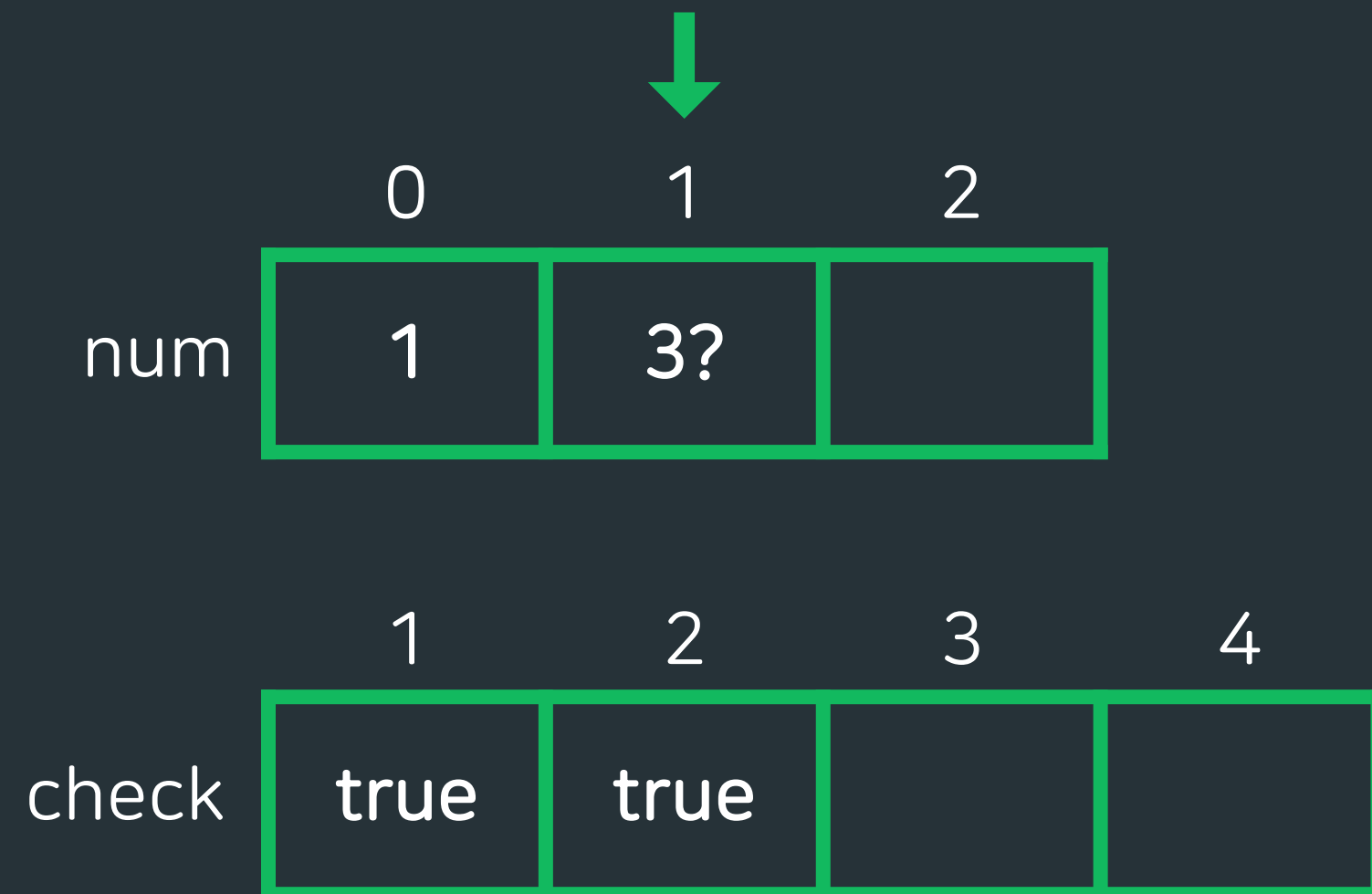
# N과 M(1)

- 1부터  $n$ 까지의 자연수 중 중복없이  $m$ 개를 고른 수열을 모두 구하는 문제
- $n = 4, m = 2$



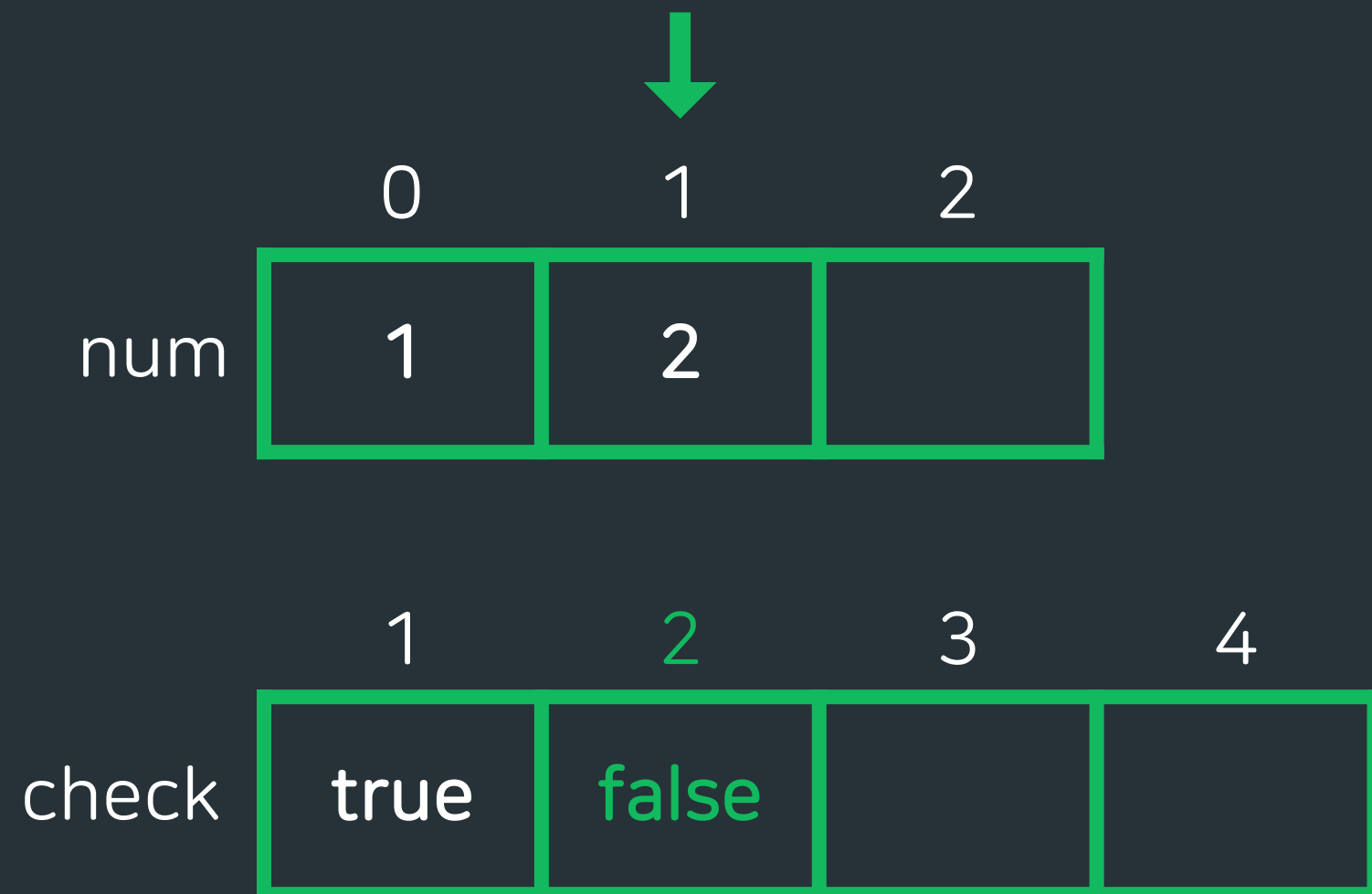
# N과 M(1)

- 1부터  $n$ 까지의 자연수 중 **중복없이**  $m$ 개를 고른 수열을 모두 구하는 문제
- $n = 4, m = 2$



# N과 M(1)

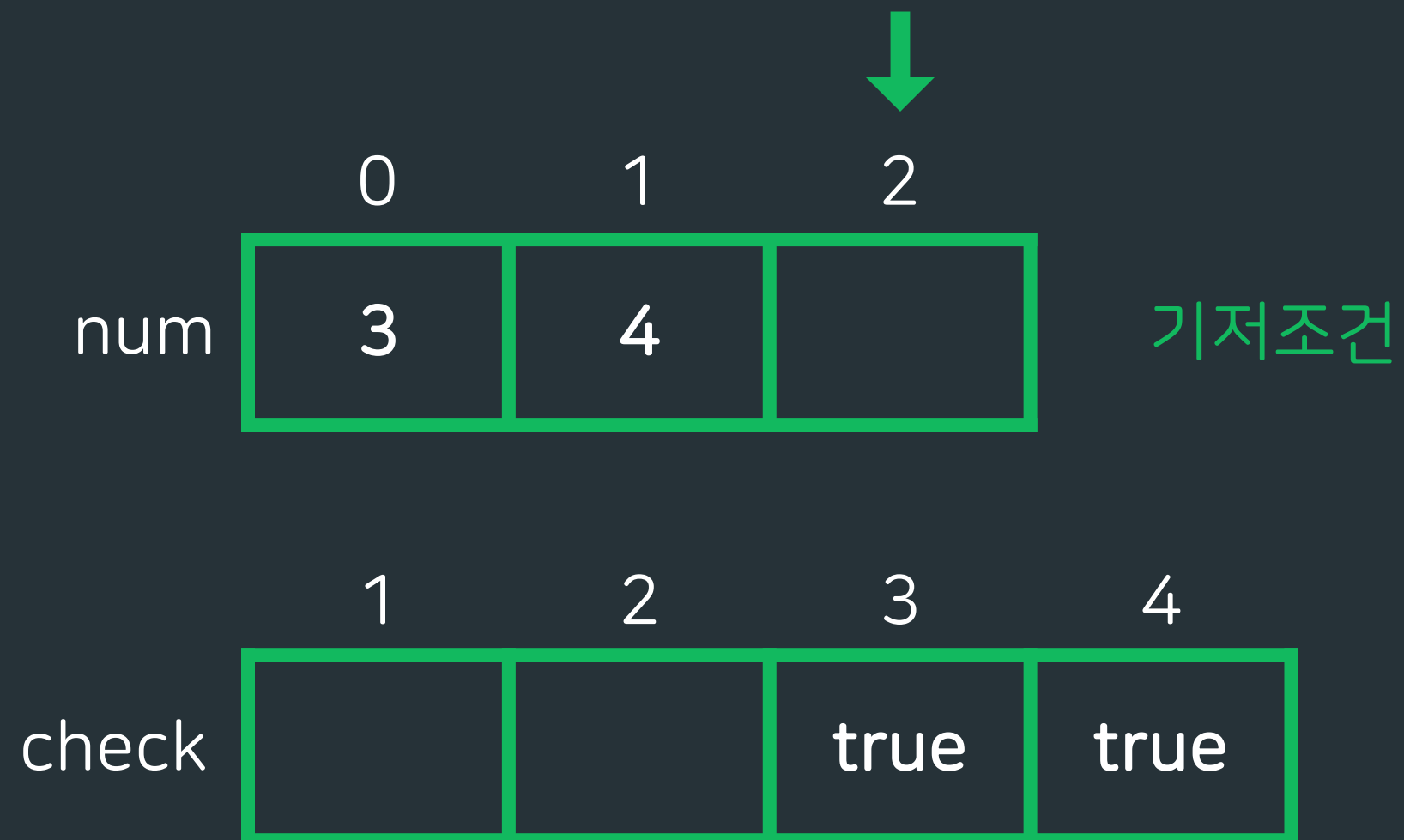
- 1부터  $n$ 까지의 자연수 중 중복없이  $m$ 개를 고른 수열을 모두 구하는 문제
- $n = 4, m = 2$



바로 탐색 이어서하지 않고, 꼭 원래 상태로 돌려놓아야 함  
그래야 나중에 해당 요소 재탐색 가능

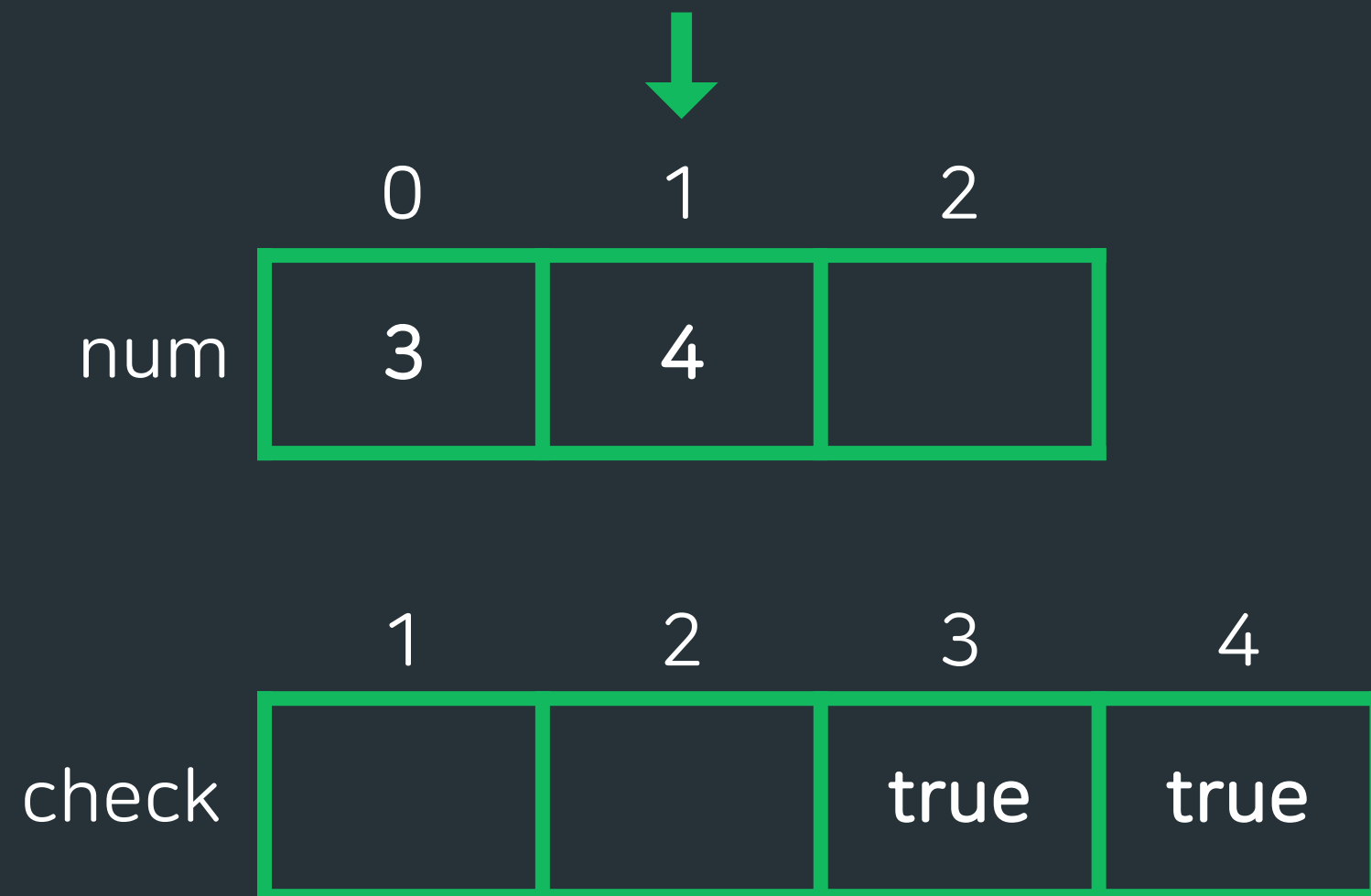
# N과 M(1)

- 1부터  $n$ 까지의 자연수 중 중복없이  $m$ 개를 고른 수열을 모두 구하는 문제
- $n = 4, m = 2$



## N과 M(1)

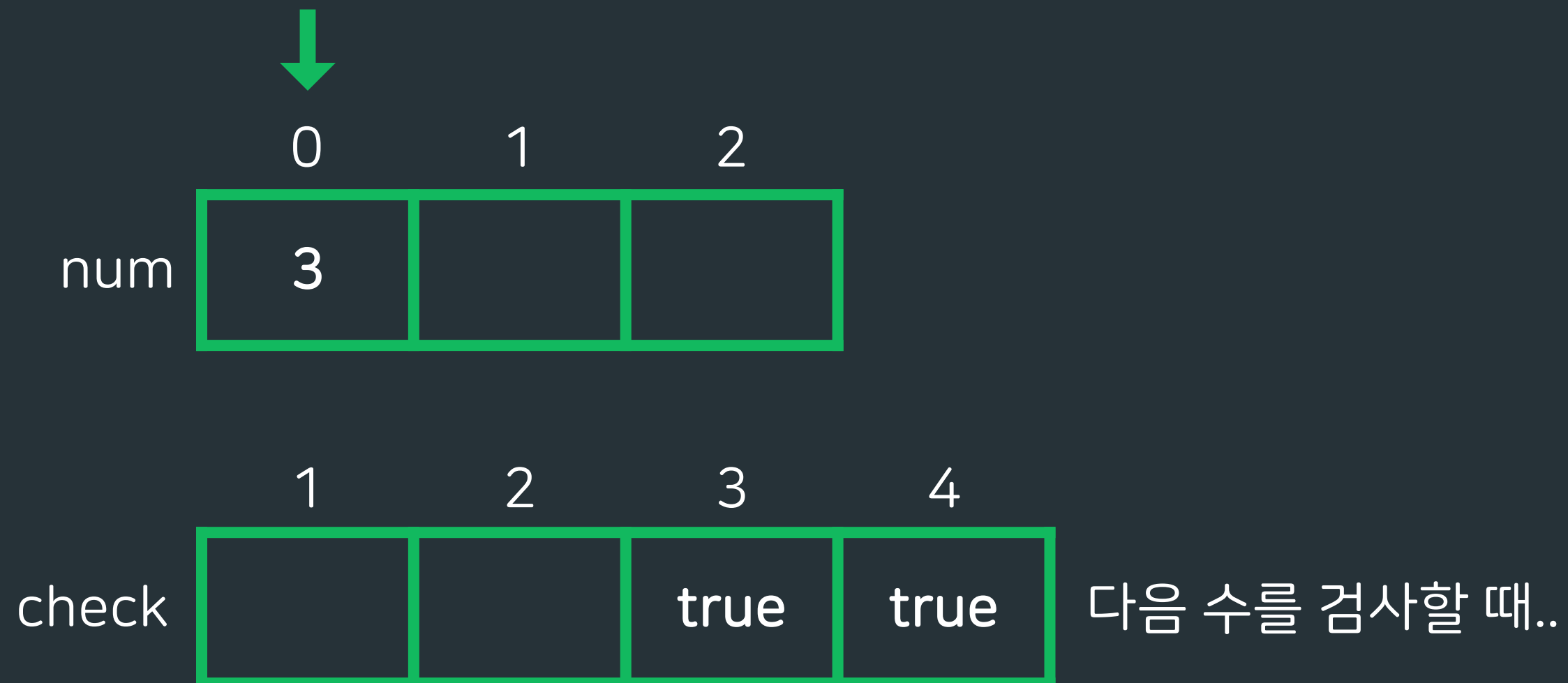
- 1부터  $n$ 까지의 자연수 중 중복없이  $m$ 개를 고른 수열을 모두 구하는 문제
- $n = 4, m = 2$



만약 원래 상태로 되돌려 놓지 않는다면?

# N과 M(1)

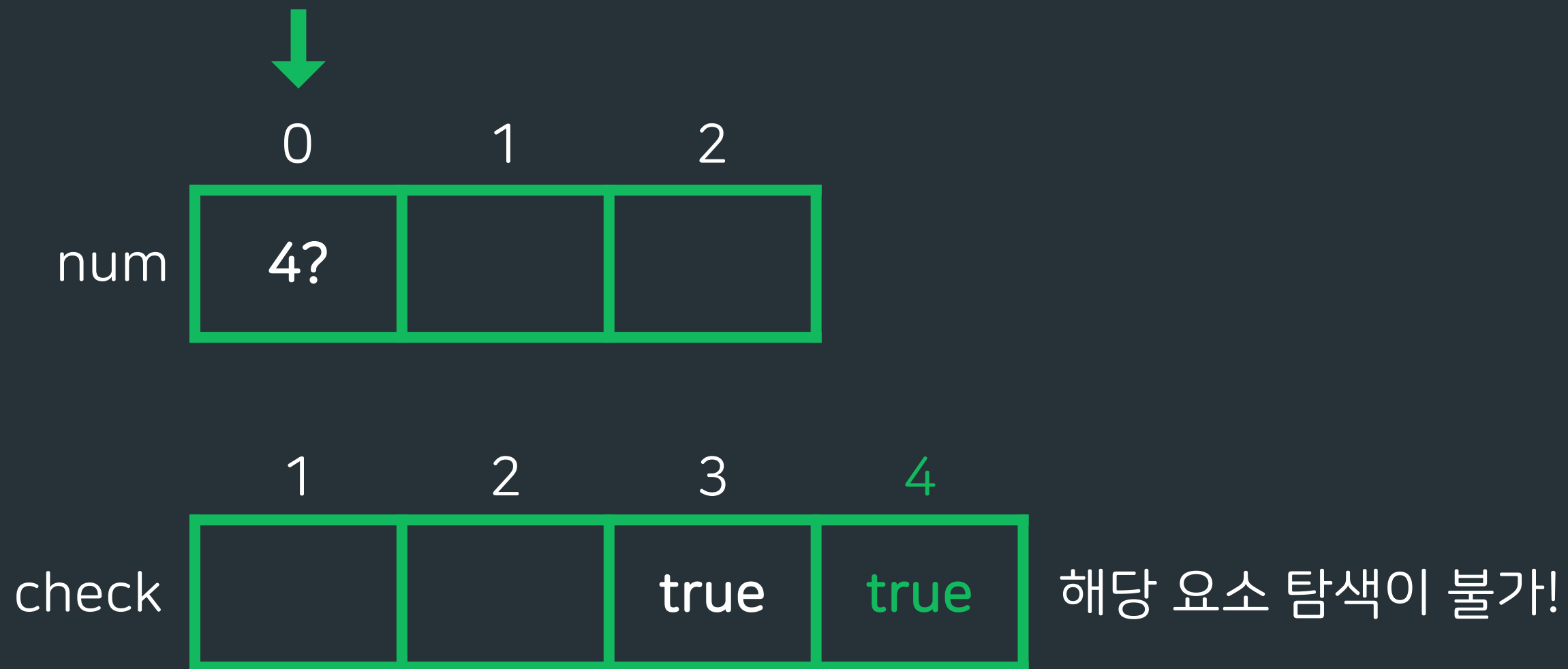
- 1부터 n까지의 자연수 중 중복없이 m개를 고른 수열을 모두 구하는 문제
- $n = 4, m = 2$





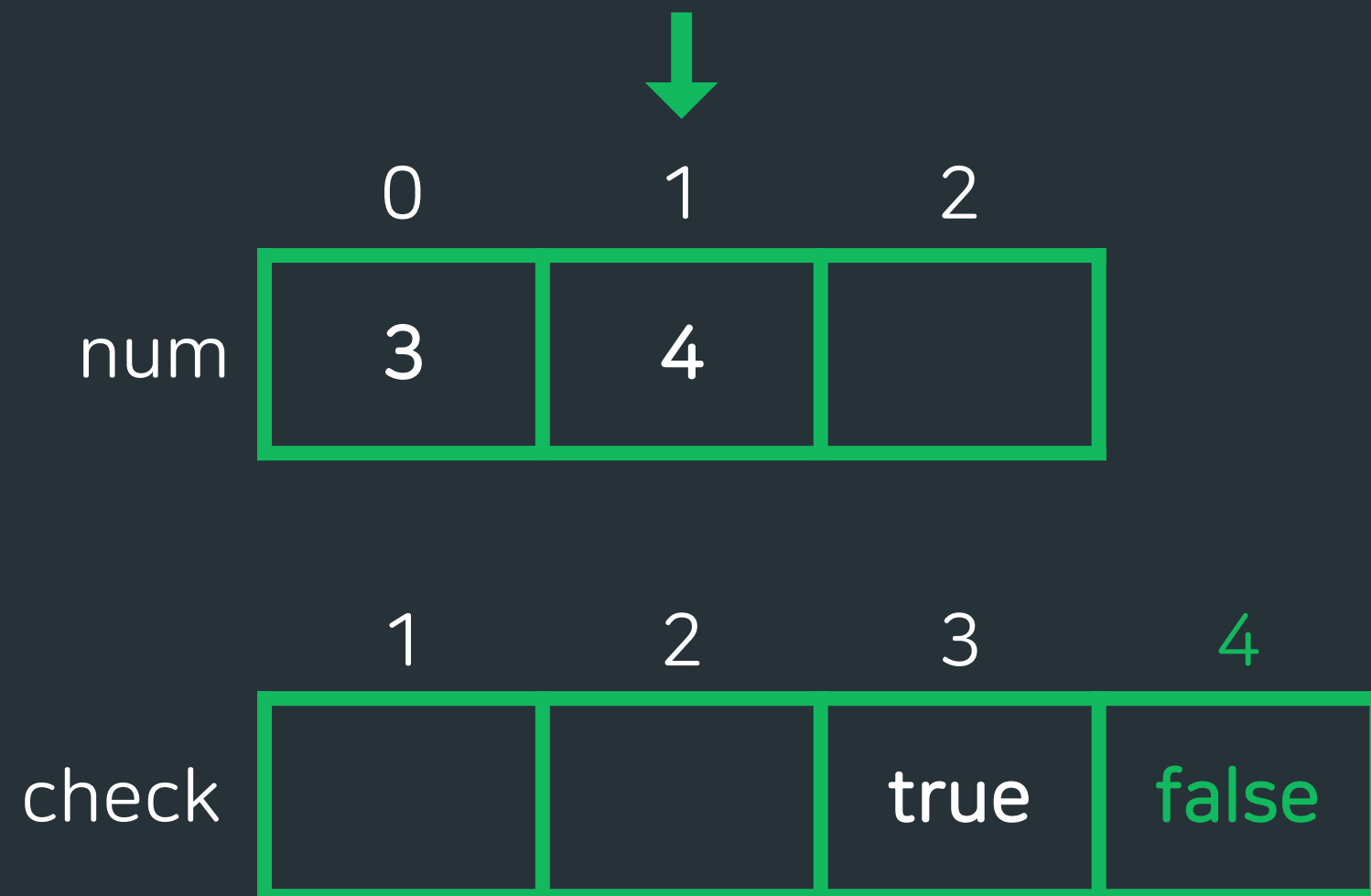
# N과 M(1)

- 1부터 n까지의 자연수 중 중복없이 m개를 고른 수열을 모두 구하는 문제
- $n = 4, m = 2$



## N과 M(1)

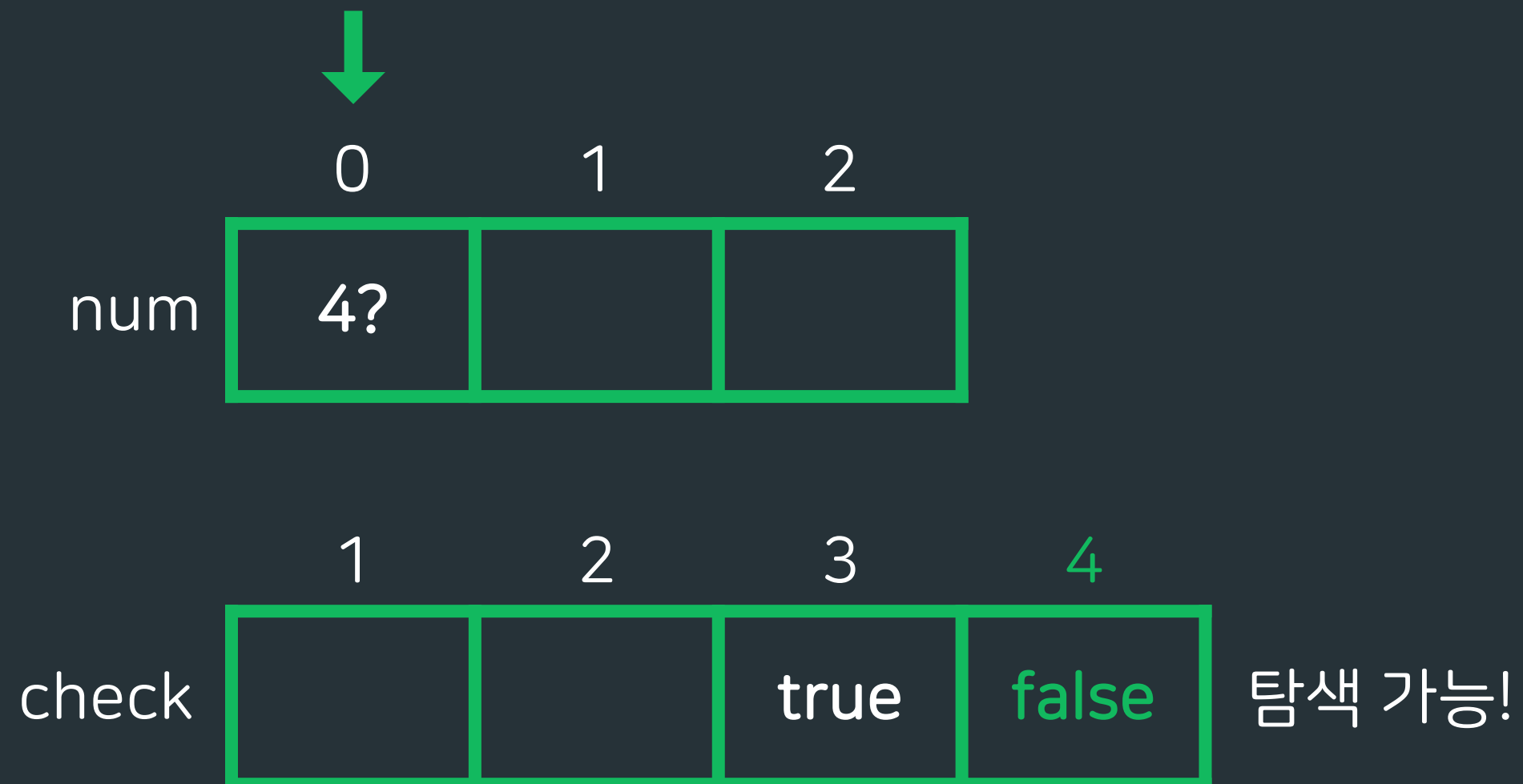
- 1부터  $n$ 까지의 자연수 중 중복없이  $m$ 개를 고른 수열을 모두 구하는 문제
- $n = 4, m = 2$



따라서 탐색을 하기 전,  
꼭 원래 상태로 돌려놓는 것이 중요

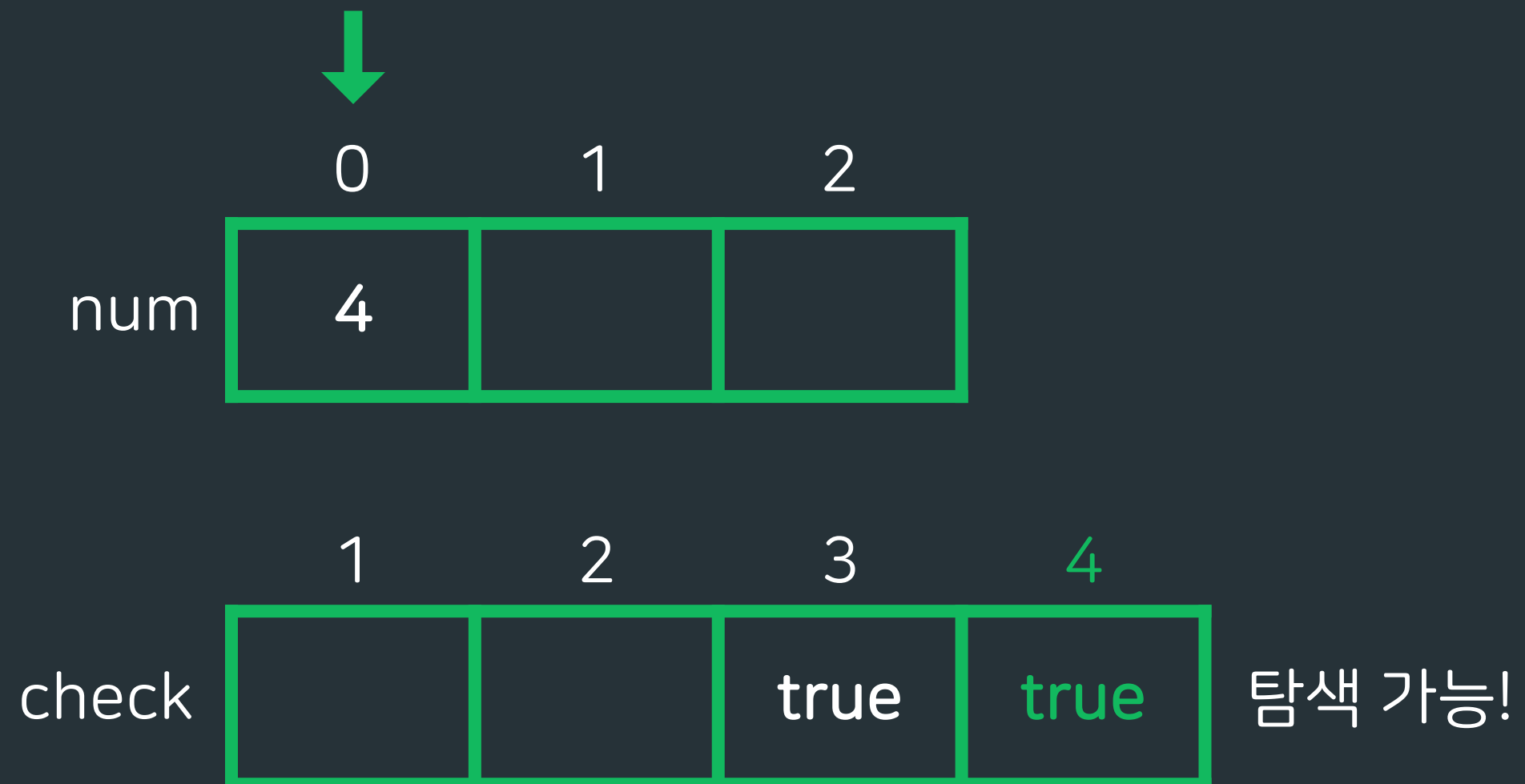
# N과 M(1)

- 1부터  $n$ 까지의 자연수 중 중복없이  $m$ 개를 고른 수열을 모두 구하는 문제
- $n = 4, m = 2$



## N과 M(1)

- 1부터  $n$ 까지의 자연수 중 중복없이  $m$ 개를 고른 수열을 모두 구하는 문제
- $n = 4, m = 2$



## /<> 15650번 : N과 M(2) – Silver 3

### 문제

- 자연수  $n$ ,  $m$ 이 주어짐
- 1부터  $n$ 까지 자연수 중 중복없이  $m$ 개를 고른 수열을 모두 구하는 문제
- 단, 수열은 오름차순임

### 제한 사항

- $n$ 의 범위는  $1 \leq m \leq n \leq 8$

## 예제 입력

4 2

## 예제 출력

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

## 기존 접근

- 제약 조건을 살펴보자  
→ 중복 x, 수열 길이가 m
- 재귀함수를 설계해보자  
→ 각 수를 넣을 때, 이미 수열 내에 있으면  
넘어감 (체크해줄 무언가가 필요)  
→ 기저조건은 길이가 m일 때!

## 추가된 접근

- 수열은 오름차순  
→ 현재 인덱스의 원소를 확인하여  
(해당 원소 + 1) 부터 다음  
인덱스의 원소 탐색 시작
- 체크해줄 무언가가..필요한가?

## /<> 9663번 : N-Queen - Gold 5

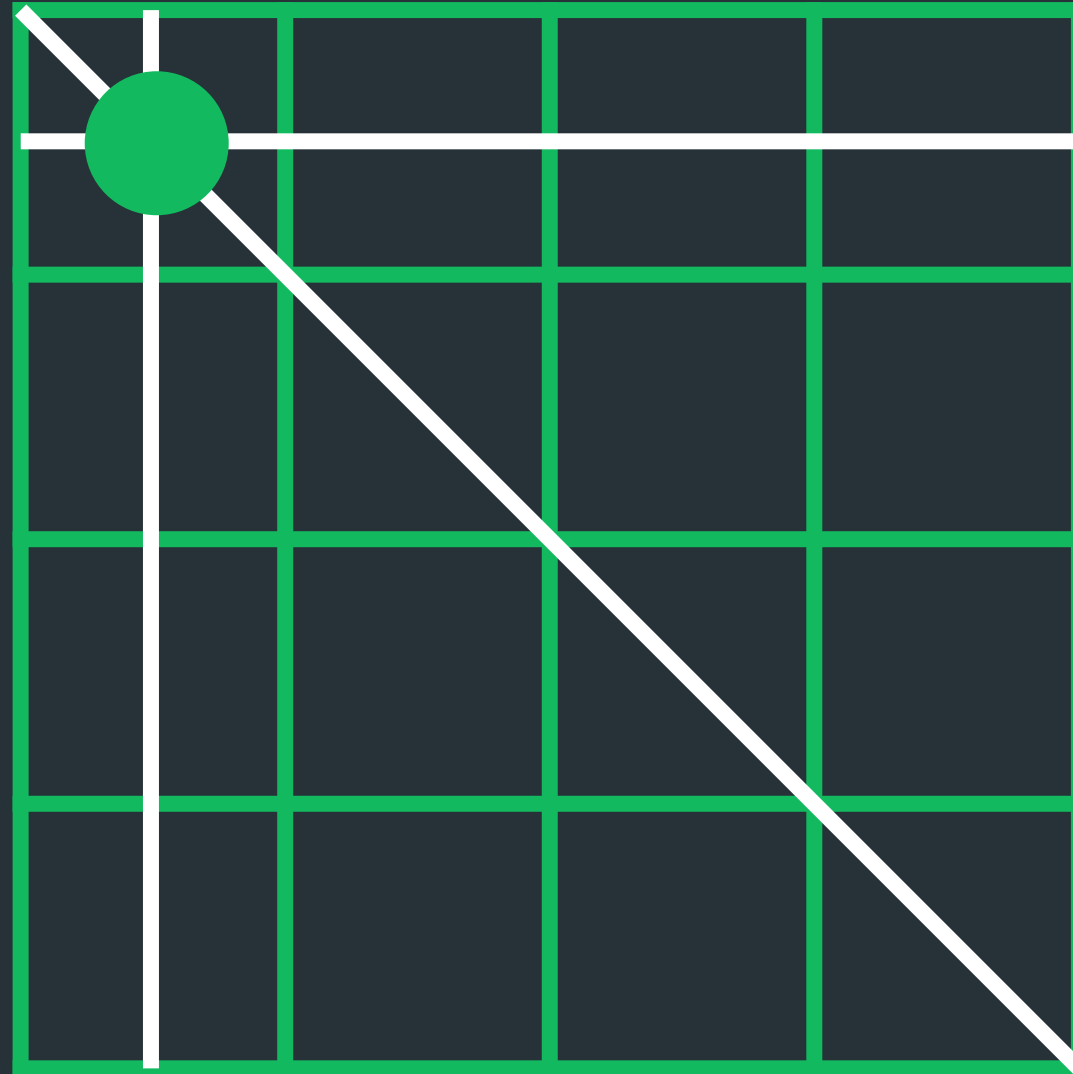
### 문제

- $N \times N$ 인 체스판 위에 퀸  $N$ 개를 서로 공격할 수 없게 놓는 경우의 수 구하는 문제

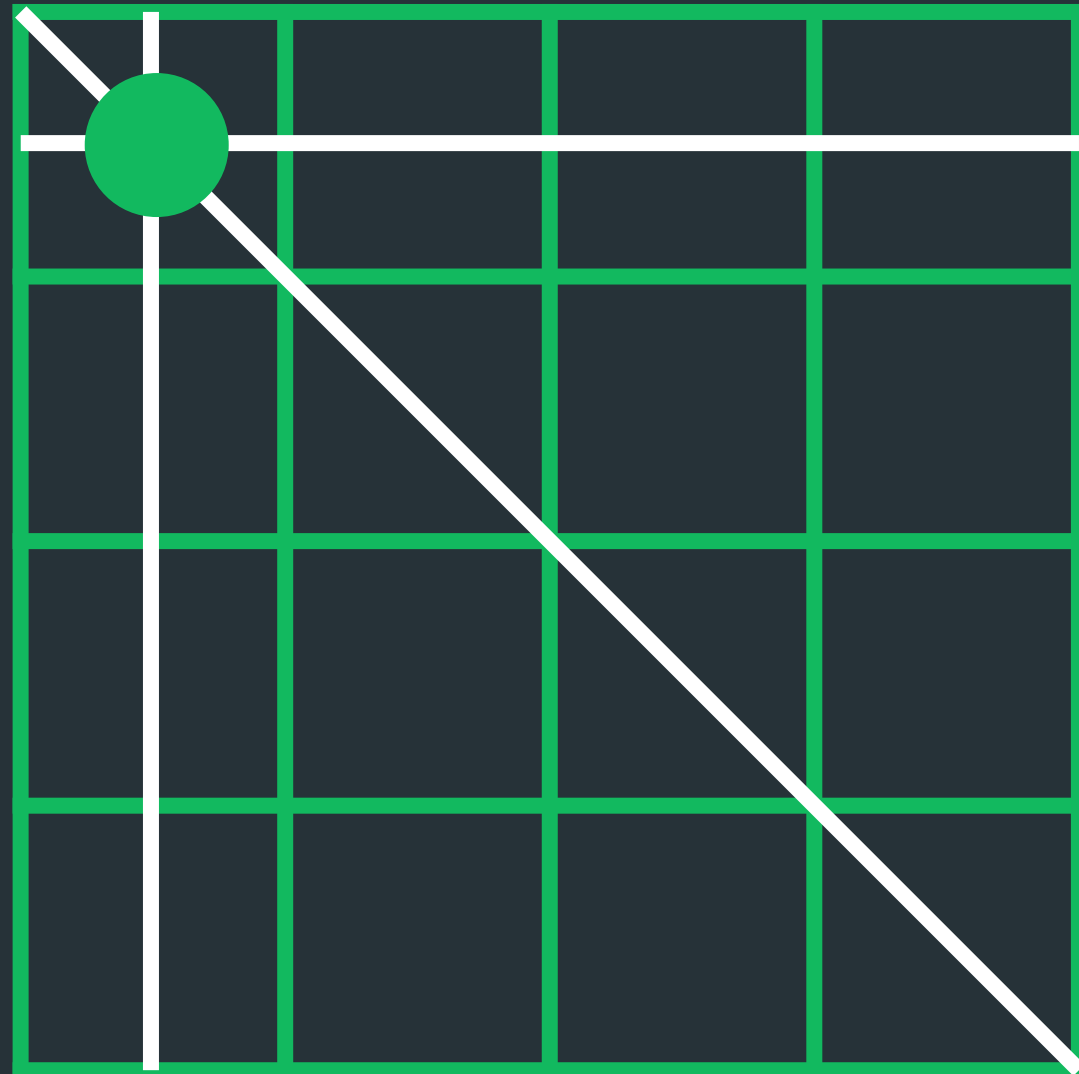
### 제한 사항

- 입력 범위는  $1 \leq N < 15$





- 퀸이 놓이면, 가로, 세로, 대각선에 위치한 곳엔 다른 퀸을 둘 수 없다.



- 퀸이 놓이면, 가로, 세로, 대각선에 위치한 곳엔 다른 퀸을 둘 수 없다.

## 제한 사항

- 입력 범위는  $1 \leq N < 15$

## 접근

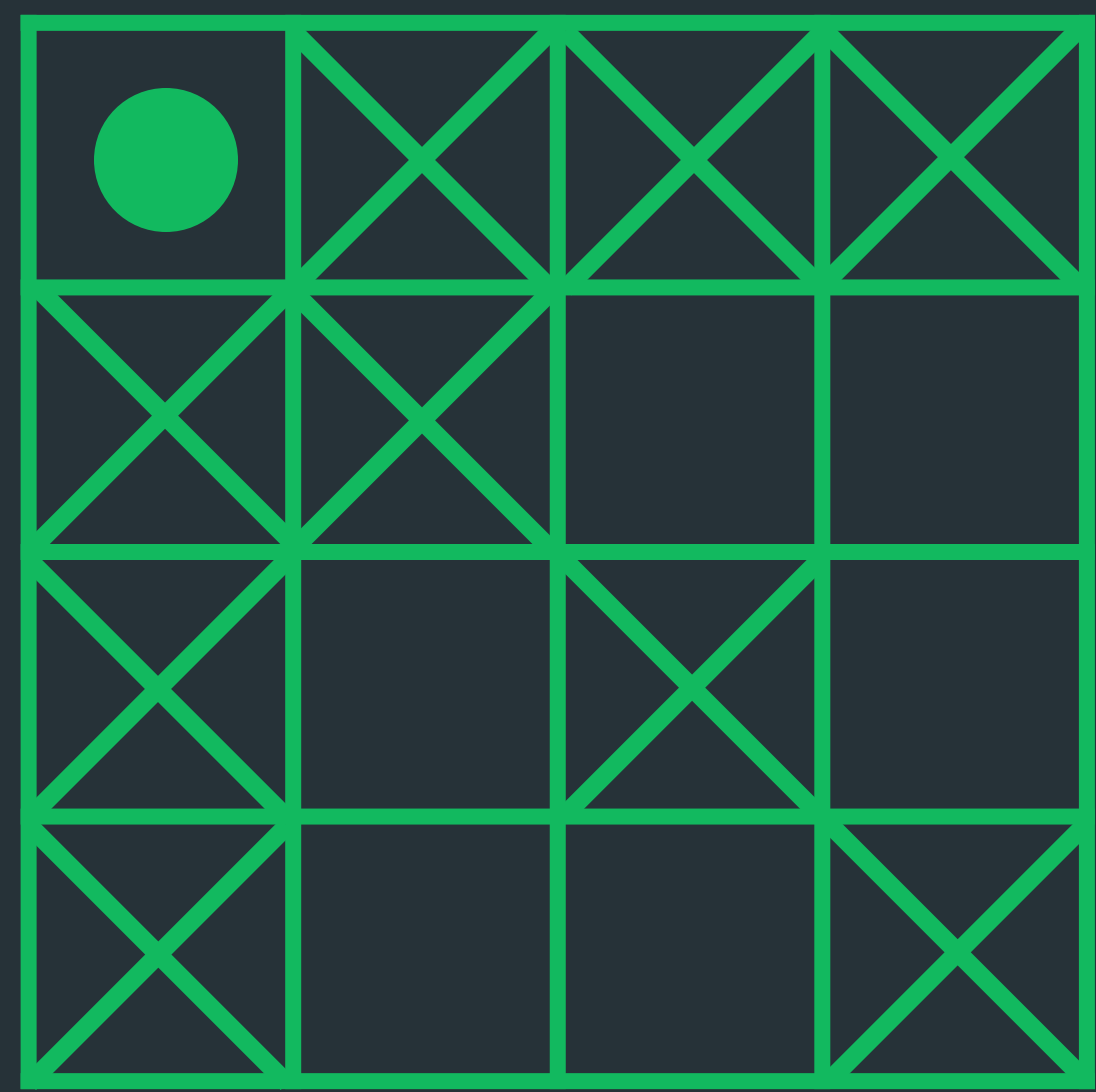
- 완전탐색?  
→  $C(255, 15) > 10^{30}$ , 절대 불가능!
- 가지치기를 할 수 있을까?  
→ 우선 세로의 중복을 피하려면 각 열마다 1개  
→ 가로의 중복을 피하려면 각 행마다 1개  
→ 즉, 가로와 세로에는 각각 1개만 들어감!

## Hint

1. N과 M문제에서 가지치기를 위해 무엇을 사용했었죠?
2. 그런데 가지치기할 곳이 1 군데는 아닌 것 같아요. 대각선도 방향이 하나가 아니죠.

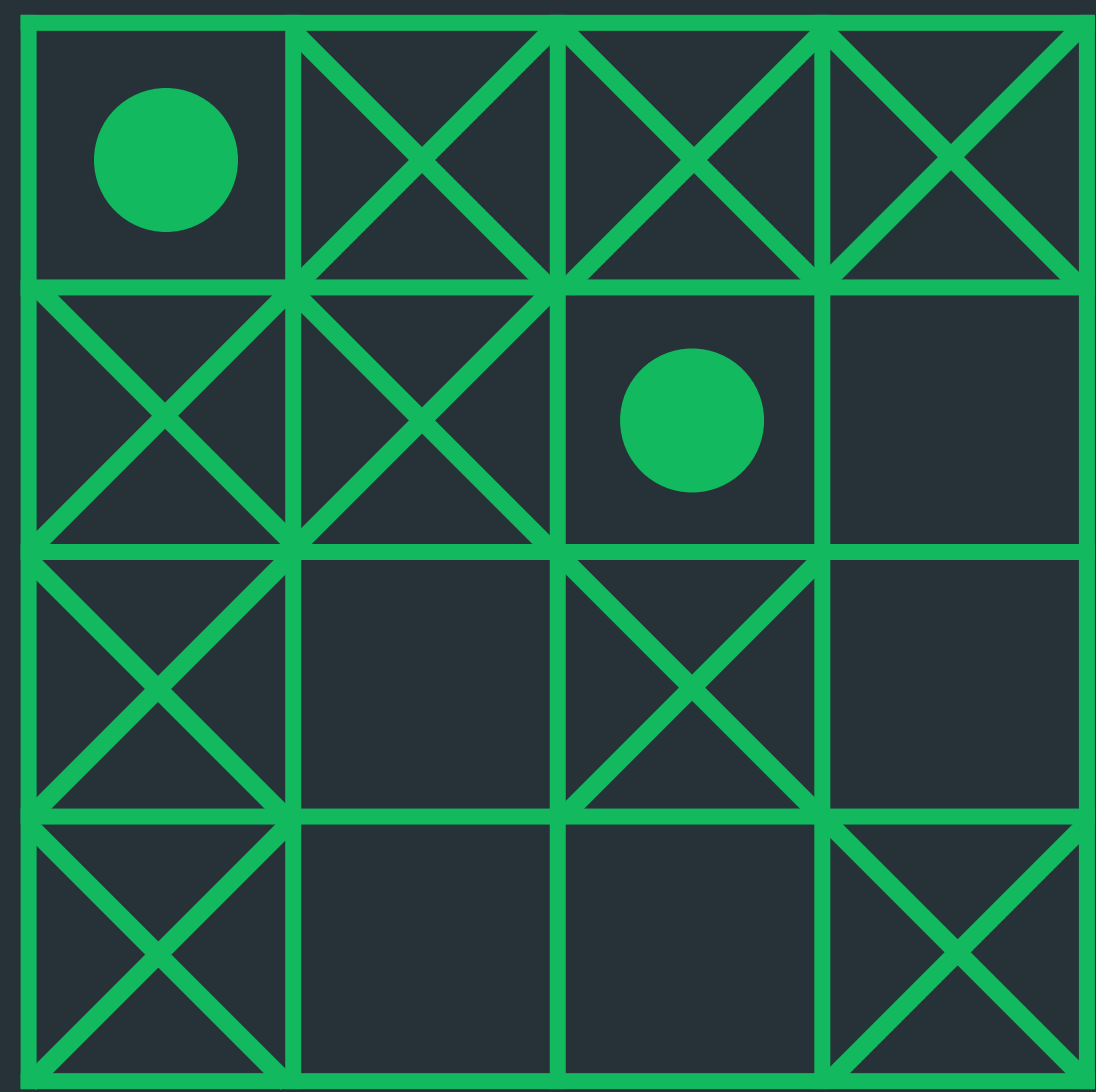
# N-Queen

● N = 4



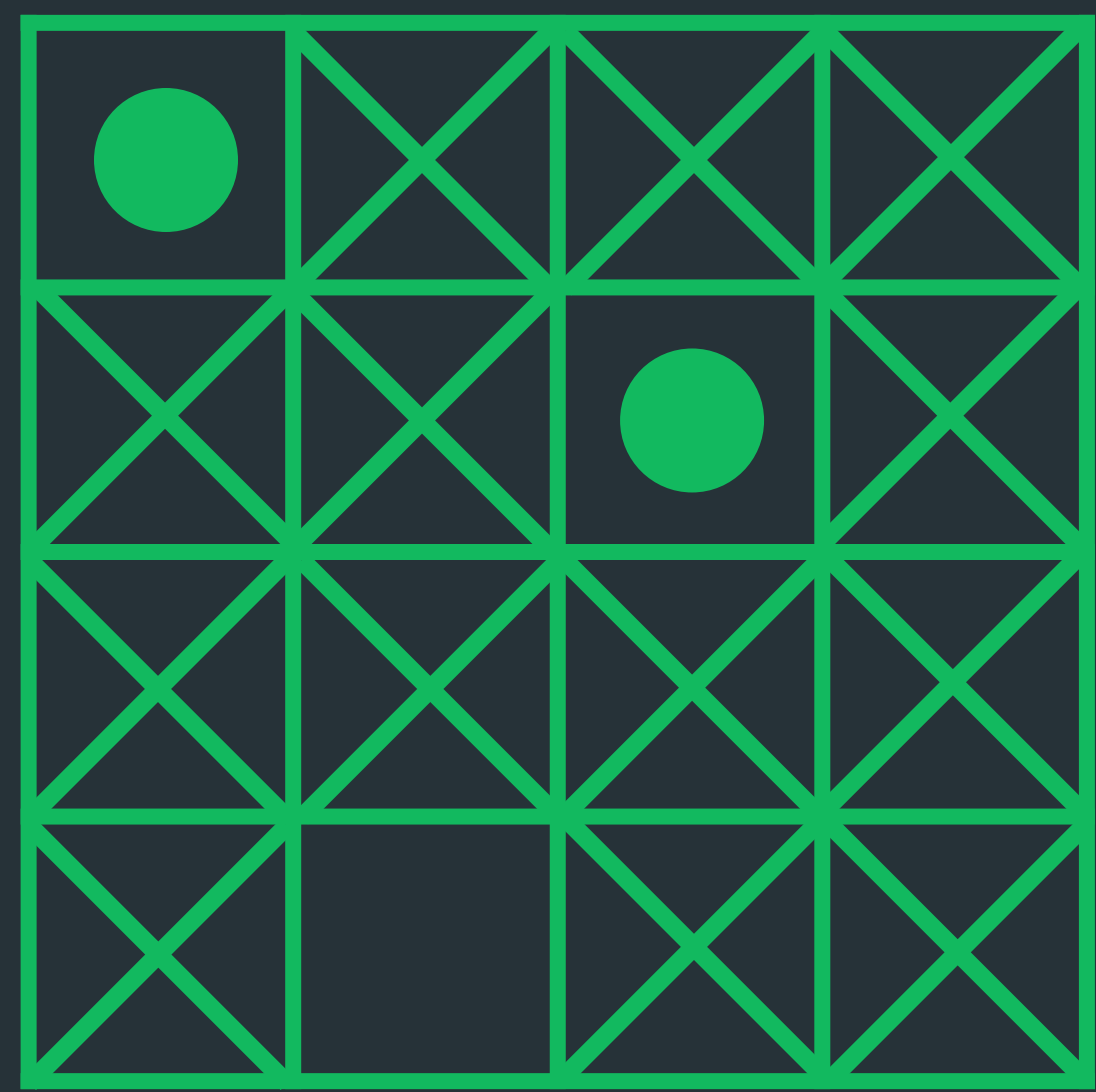
# N-Queen

● N = 4



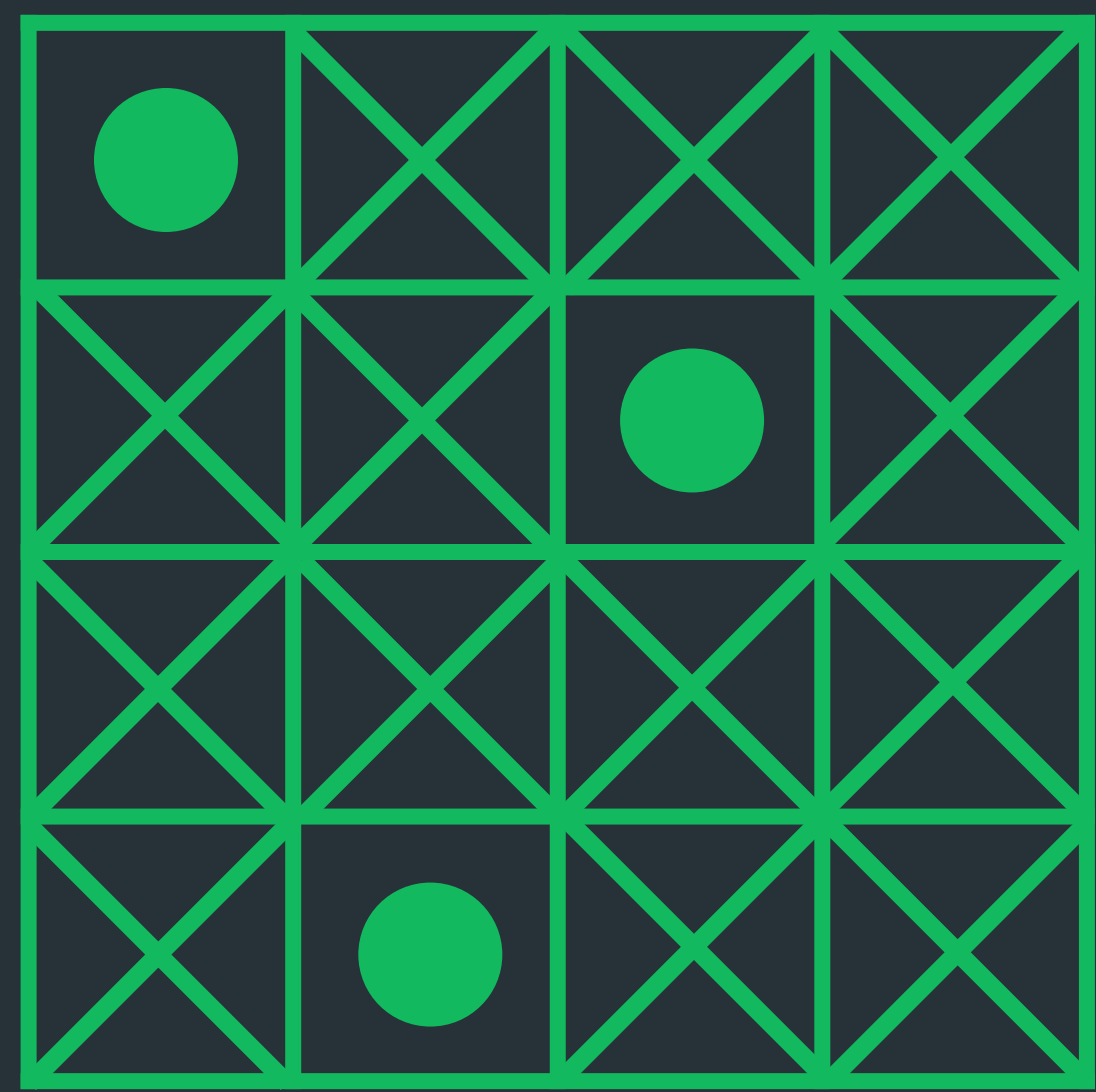
# N-Queen

● N = 4



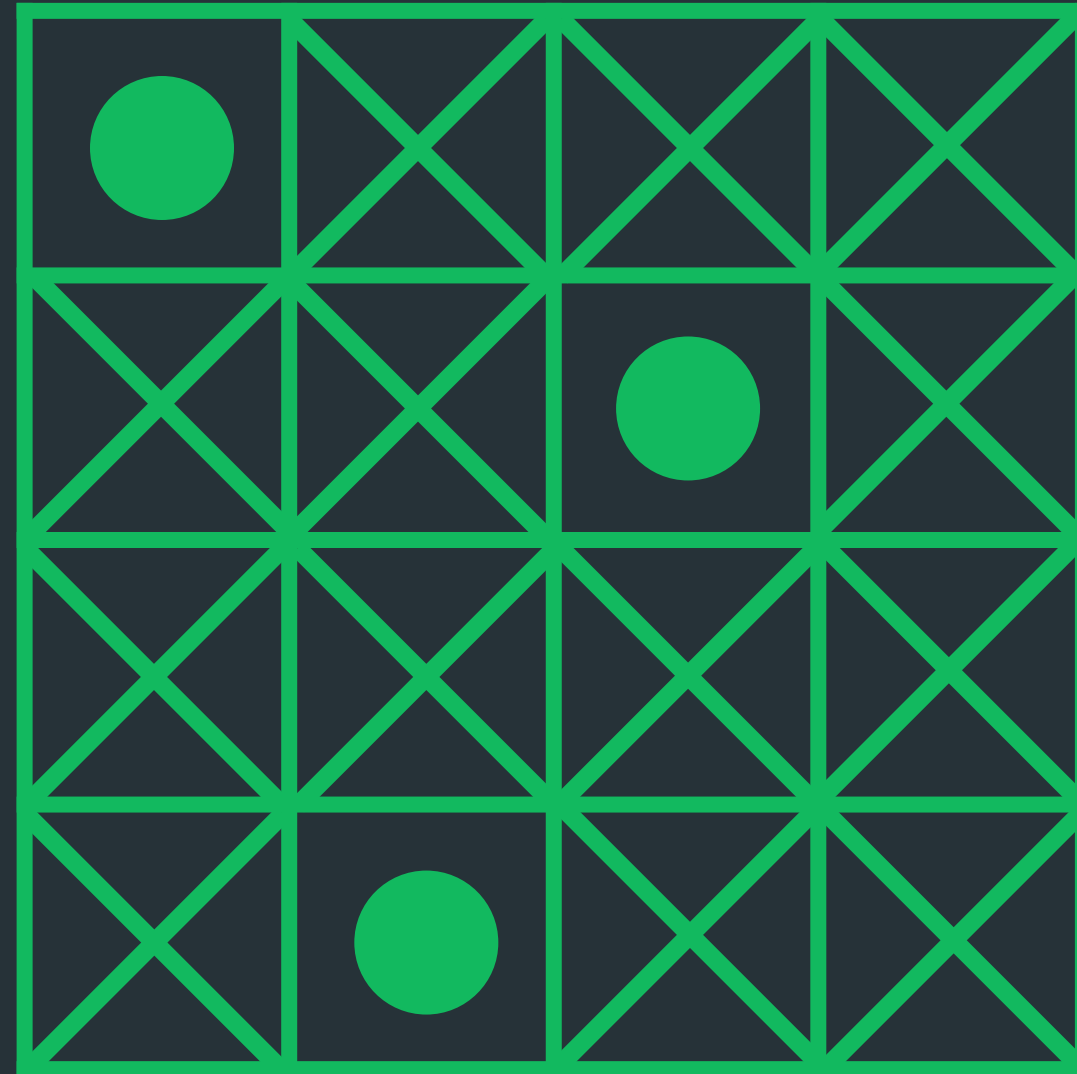
# N-Queen

● N = 4



# N-Queen

● N = 4

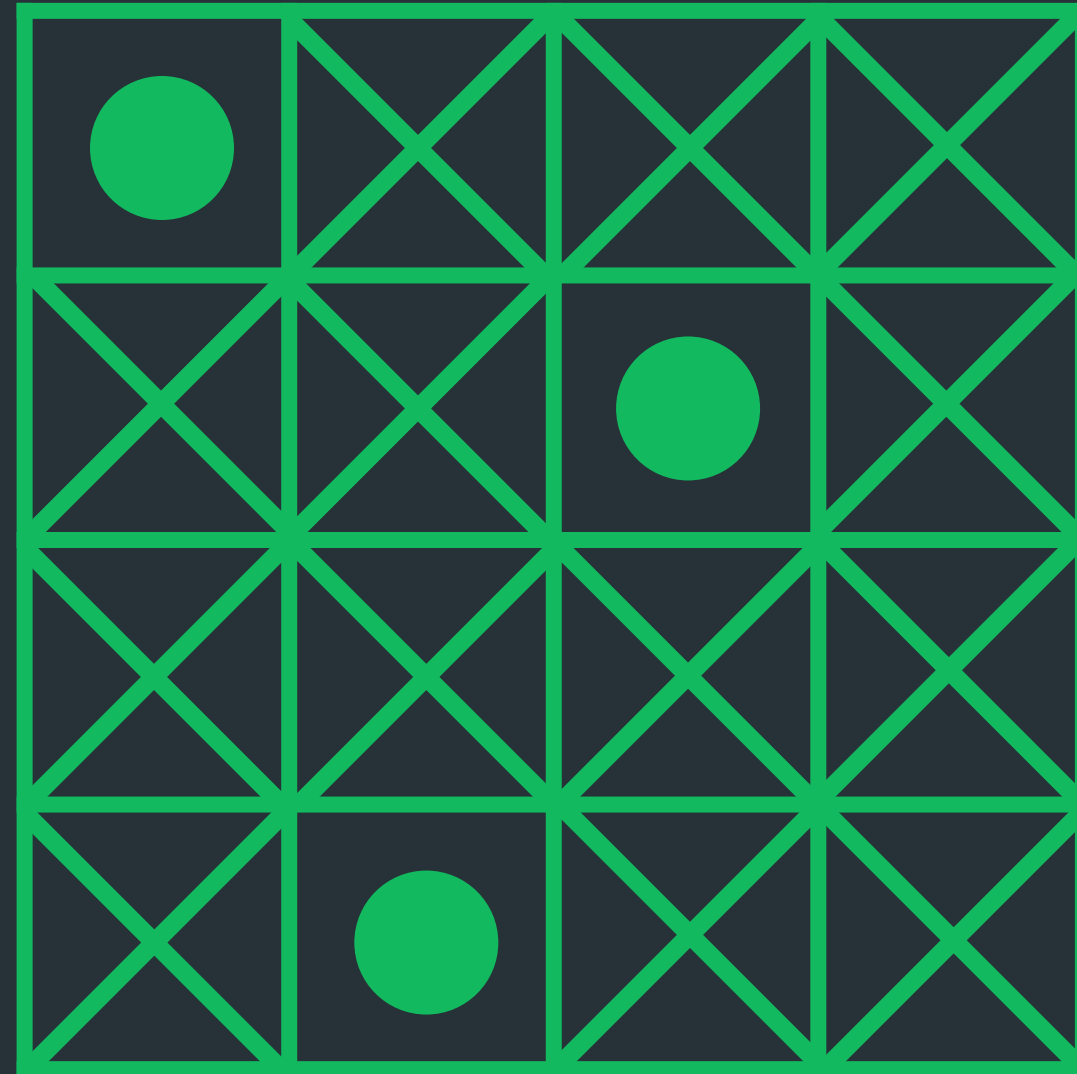


● 더 이상 탐색 불가! 더불어 퀸이 3개라 정답도 아님  
→ 다시 전으로 돌아감



# N-Queen

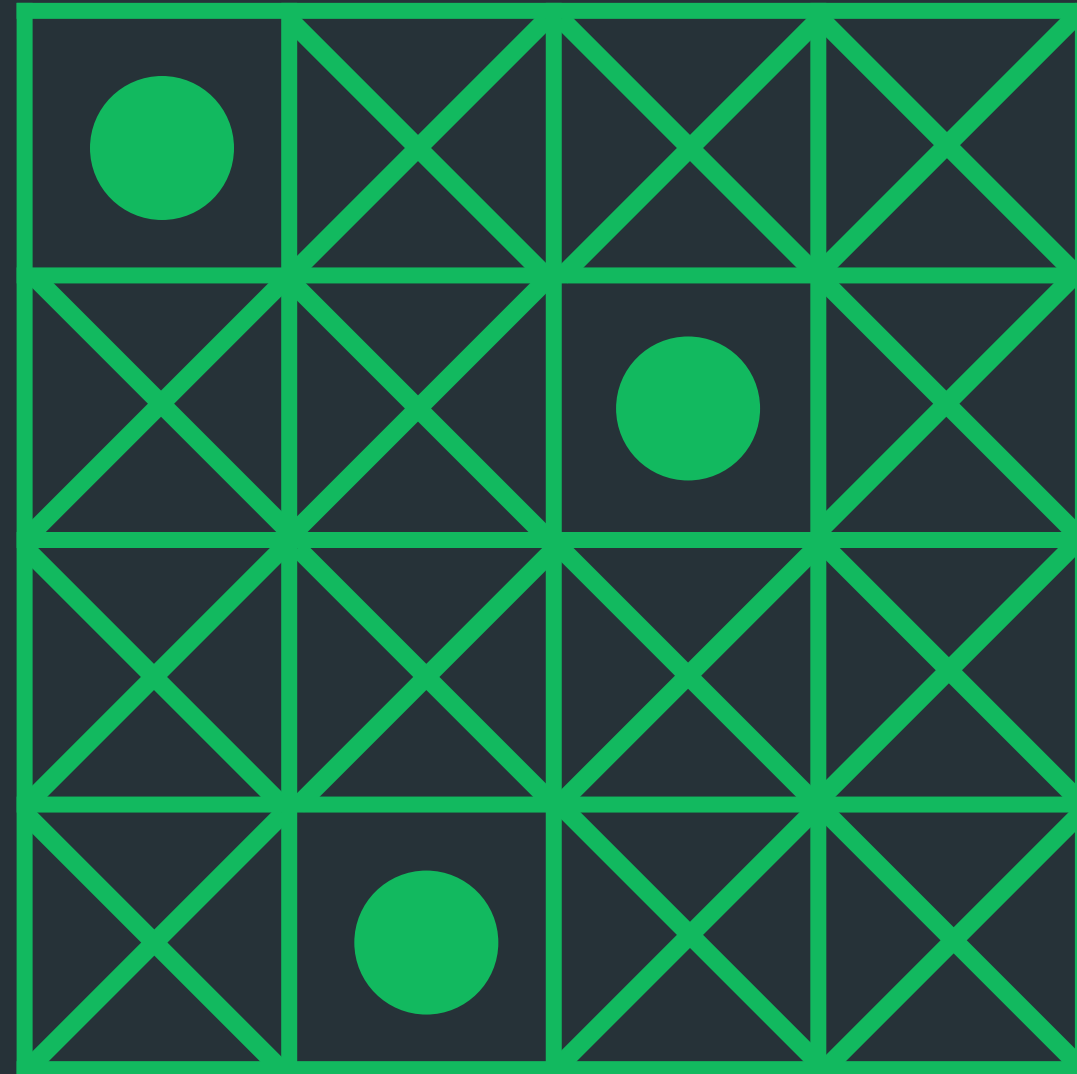
- $N = 4$



- 퀸을 놓을 수 있는 자리를 **체크배열**로 → 2차원? →  
체크를 하는 데 반복문을 사용할테므로 **시간 걸려서 안됨!**

# N-Queen

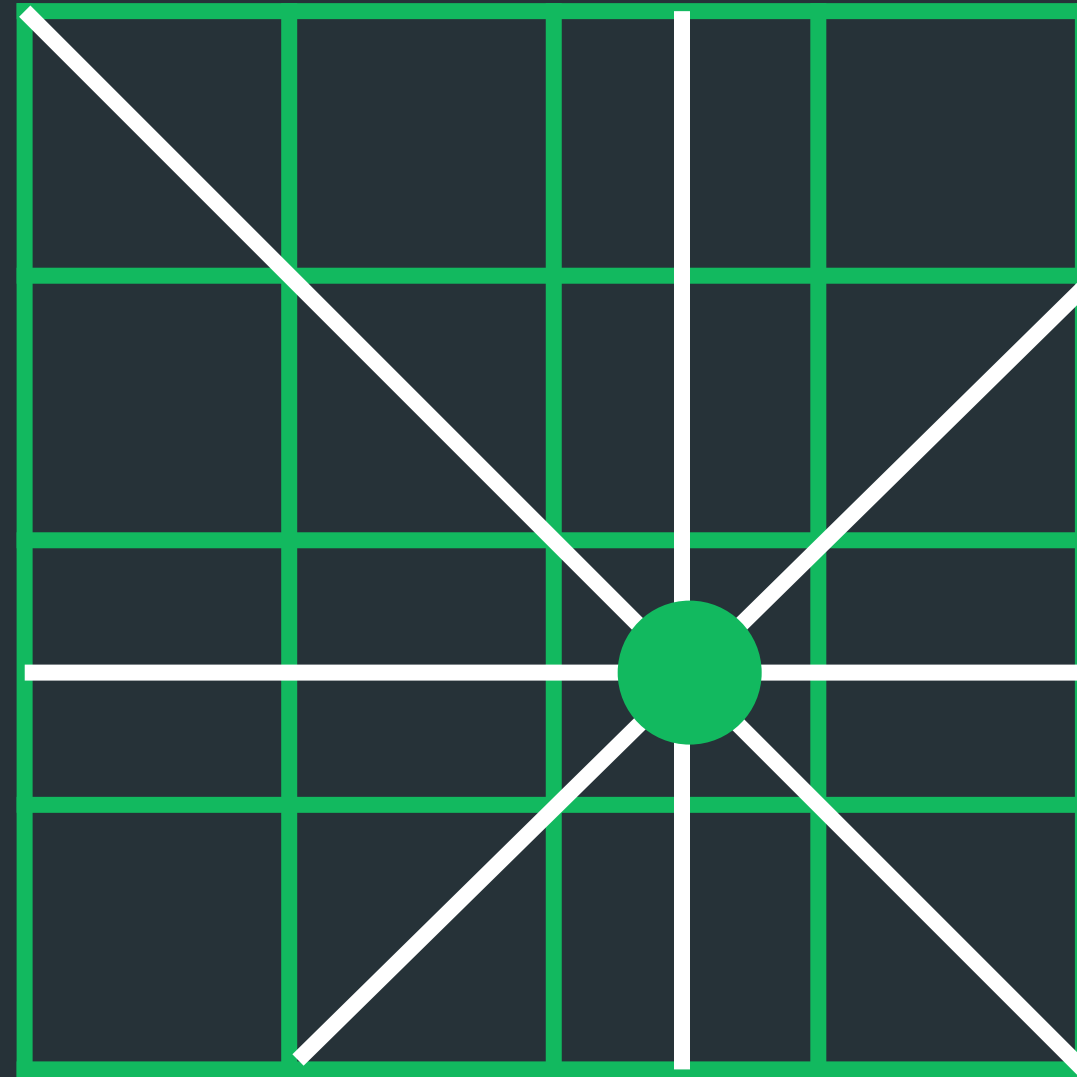
- $N = 4$



- 퀸을 놓을 수 있는 자리를 체크배열로 → 한 번에 체크할 수 있는 배열이 필요 → 각 줄을 인덱스로!

# N-Queen

● N = 4



- 퀸이 놓이면, 가로, 세로, 대각선에 위치한 곳엔 다른 퀸을 둘 수 없다.
- 가로와 세로엔 각각 1개 배치 -> 가로를 기준으로 세로 배치 체크

→ 세로, 좌하향 대각선, 우하향 대각선의 배치 가능 여부를 체크배열로 구현!

# N-Queen

- $N = 4$
- 행:  $i$ , 열:  $j$

Check[  $j$  ]

(0,0)	(0,1)	(0,2)	(0,3)
(1,0)	(1,1)	(1,2)	(1,3)
(2,0)	(2,1)	(2,2)	(2,3)
(3,0)	(3,1)	(3,2)	(3,3)

→ 세로, 좌하향 대각선, 우하향 대각선의 배치 가능 여부를 체크배열로 구현!

# N-Queen

- $N = 4$
- 행:  $i$ , 열:  $j$

0	(0,0)	(0,1)	(0,2)	(0,3)
1	(1,0)	(1,1)	(1,2)	(1,3)
2	(2,0)	(2,1)	(2,2)	(2,3)
3	(3,0)	(3,1)	(3,2)	(3,3)
	3	4	5	6

Check[i + j]

→ 세로, 좌하향 대각선, 우하향 대각선의 배치 가능 여부를 체크배열로 구현!

# N-Queen

- $N = 4$
- 행:  $i$ , 열:  $j$

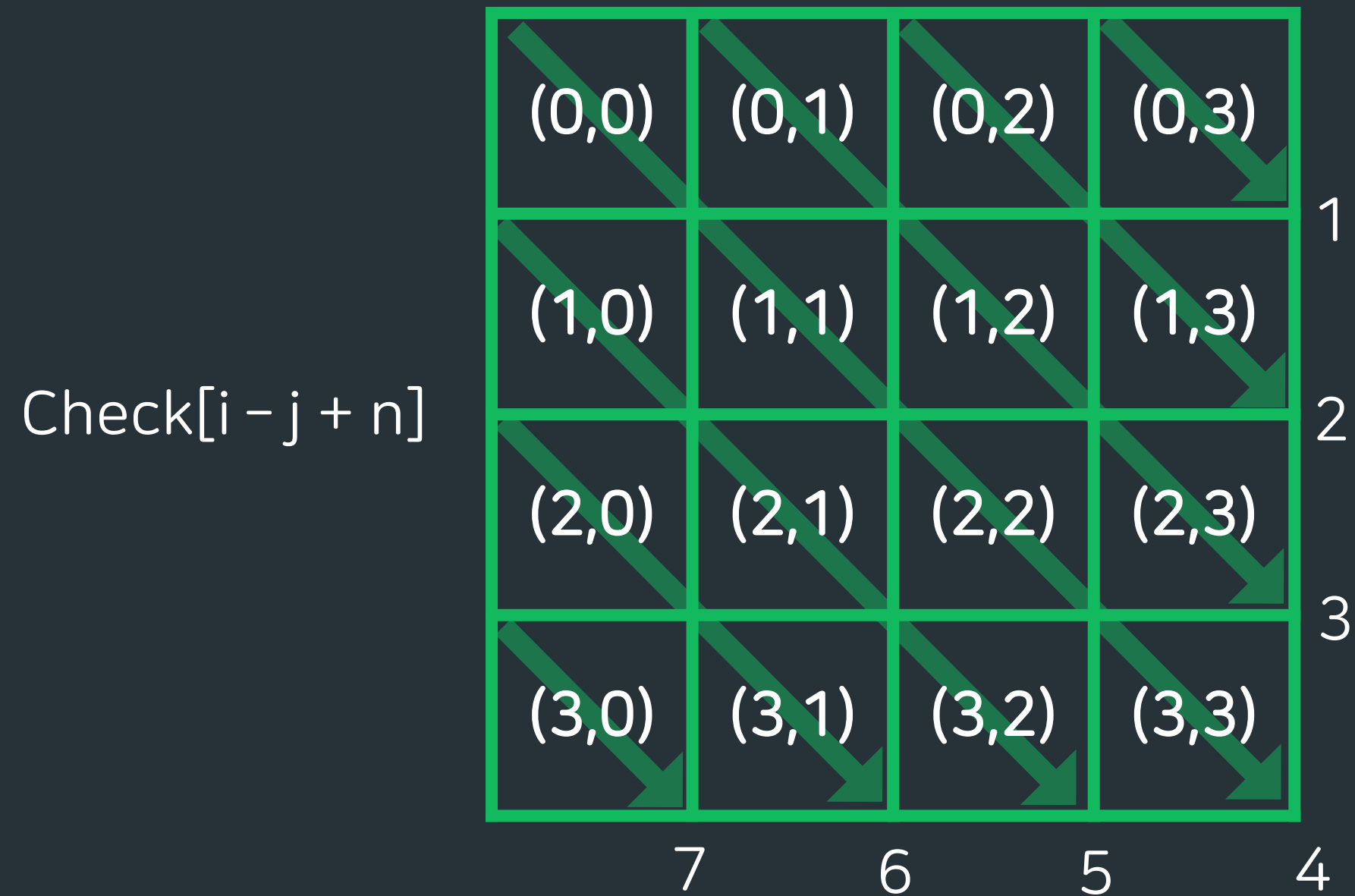
Check[ $i - j$ ]



→ 세로, 좌하향 대각선, 우하향 대각선의 배치 가능 여부를 체크배열로 구현!

# N-Queen

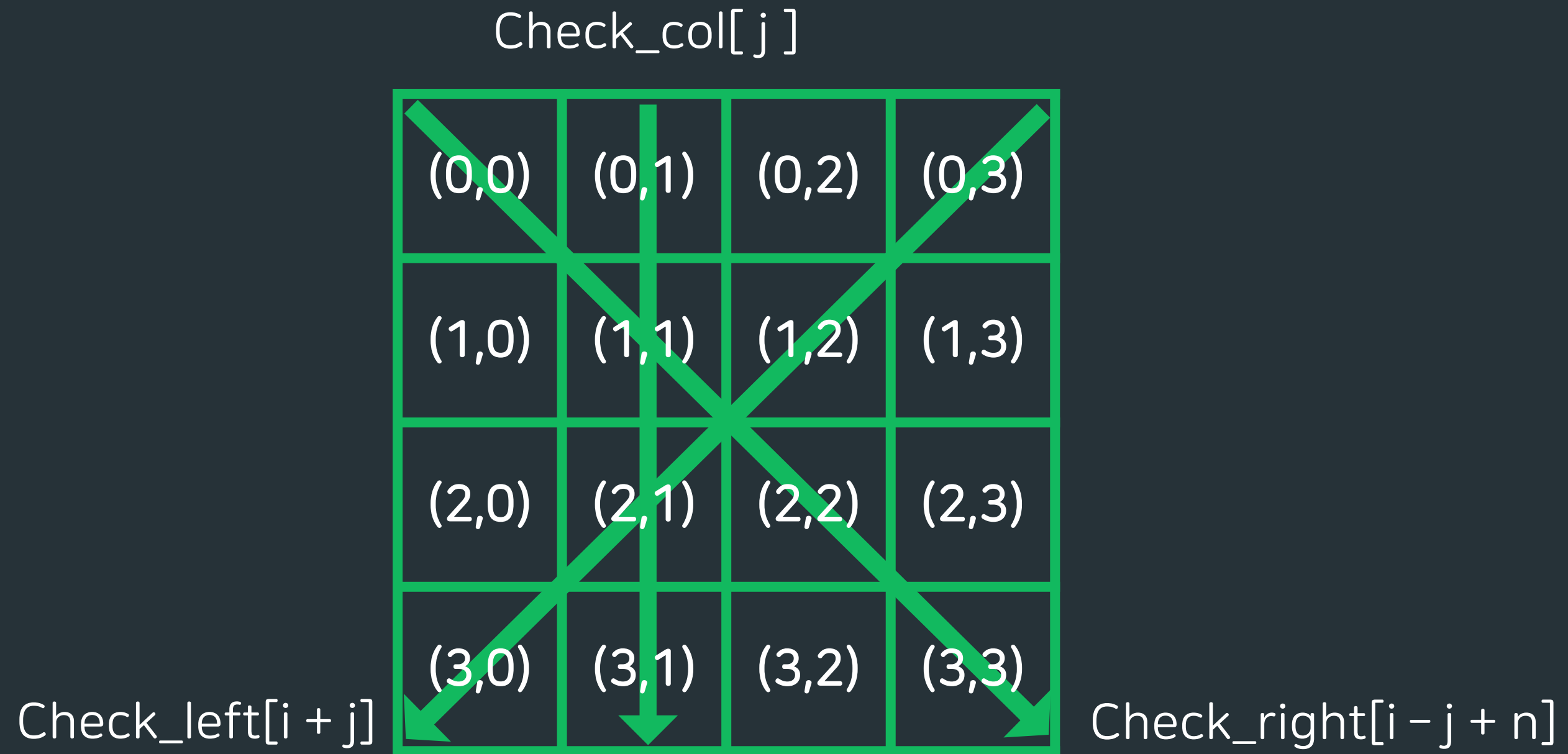
- $N = 4$
- 행:  $i$ , 열:  $j$



→ 세로, 좌하향 대각선, 우하향 대각선의 배치 가능 여부를 체크배열로 구현!

# N-Queen

- $N = 4$
- 행:  $i$ , 열:  $j$



→ 세로, 좌하향 대각선, 우하향 대각선의 배치 가능 여부를 체크배열로 구현!



# N-Queen

- $N = 4$
- 행:  $i$ , 열:  $j$
- (ex)  $(1, 1)$ 에 퀸을 놓는다면

Check\_col[1] = true

Check\_left[2] = true

(0,0)	(0,1)	(0,2)	(0,3)
(1,0)	(1,1)	(1,2)	(1,3)
(2,0)	(2,1)	(2,2)	(2,3)
(3,0)	(3,1)	(3,2)	(3,3)

Check\_right[4] = true

5 9663	맞았습니다!!	2020 KB	1684 ms	← 현재 풀이
5 9663	맞았습니다!!	2020 KB	4528 ms	← 다른 풀이

→ 가지치기를 어떻게 하느냐에 따라 백트래킹 속도 효율이 달라진다!

\*해당 다른 풀이는 깃허브 라이브 코딩 폴더에 별도로 올라갈 예정입니다!

\*가지치기가 어떻게 다른지 비교해보아요.

# 백트래킹 문제는 다른 풀이가 가능한 경우가 많아요

## 비트마스킹

- 비트(0 / 1)의 연산을 활용한 알고리즘
- 배열의 원소 상태가 2가지로 나뉠 수 있는 백트래킹 문제에서 활용 가능
- 그러나! 백트래킹이 아닌 완전탐색이므로 속도는 백트래킹보다 느림

## permutation

- 다음 순열(next\_permutation) 혹은 이전 순열(prev\_permutation)을 구하는 알고리즘
- 원소의 순서를 정해야 하는 백트래킹 문제에서 활용 가능
- 그러나! 백트래킹이 아닌 완전탐색이므로 속도는 백트래킹보다 느림

백트래킹 문제는 다른 풀이가 가능한 경우가 많아요



따라서, 이번 과제는 백트래킹 알고리즘뿐만 아니라 정말 다양한 풀이와 설명(비트마스킹, permutation, etc.)으로 답안을 준비했으니 꼭 확인해주세요!

## 정리

- 완전탐색에서 나아가 특정한 조건을 만족하는 경우만 탐색하는 백트래킹
- 입력 범위가 작고(보통 20을 넘지 않는다), 재귀함수로 주로 구현!
- 백트래킹을 쉽게 구현하다보니, 전역변수와 void 함수를 사용하지만 다른 알고리즘에선 가능한 쓰지 않는게 좋다
- 어디서 가지치기를 해야하는지와 어떻게 할지 파악하는게 중요
- 가지치기를 얼마나 잘하냐에 따라 시간의 효율도 높아짐

## 이것도 알아보세요!

- 문제를 풀고 제출한 후, 다른 사람들의 풀이와 내 풀이의 시간을 비교해보아요. (가지치기 효율)
- next\_permutation과 비트마스킹으로 백트래킹 문제를 풀어보고 시간을 비교해보아요.

## 3문제 이상 선택

- /<> 15661번 : 링크와 스타트- Silver 1
- /<> 15665번 : N과 M(11) - Silver 2
- /<> 2529번 : 부등호 - Silver 2
- /<> 1759번 : 암호 만들기 - Gold 5
- /<> 2580번 : 스도쿠 - Gold 4
- /<> 10971번 : 외판원 순회 2 - Silver 2