알튜비튜
백트래킹

오늘은 코딩테스트에 많이 나오는 알고리즘 중 하나인 백트래킹에 대해 배웁니다.
전번에 배운 완전탐색(브루트포스)에서 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

복습

과정

- 어떤 노드의 유망성(promising)을 점검
  → 조건에 맞는지 안맞는지
  → 답이 될 수 있는지 없는지
- 유망하지 않다면(non-promising) 배제함 (가지치기)

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

도전 문제 1

#알튜비튜

프로그래머스 : 소수 찾기 Lv.2

문제

- 0~9가 적힌 여러 개의 종이 조각이 존재
- 각 종이 조각에 적힌 숫자가 적힌 문자열 numbers가 주어질 때
- 조각들을 이어 붙여 만들 수 있는 소수의 개수를 구하는 문제

제한 사항

- 주어지는 조각의 개수는 1 이상 7 이하

예제

#알튜비튜

예제 입력	예제 출력
"17"	3

소수 찾기

#알튜비튜

접근

- N과 M 문제와 유사
  → 중복 X, 대신 길이 제한이 없다

재귀 호출 과정 생각해보기

#알튜비튜

접근

- 길이가 len인 수열을 만드는 과정
  : 길이가 1에서부터 점점 늘어난다
  → 과정 중에 길이가 1, 2, …, (len-1)인 수열이 만들어진다

→ 최대 길이의 수열을 만들면서 중간 과정에서 만들어지는 수도 소수인지 체크
→ 모든 길이의 수열을 다 고려하게 됨

도전 문제 2

2580번 : 스도쿠 - Gold 4

문제

- 빈칸이 있는 스도쿠 판이 주어짐
- 모든 빈 칸이 채워진 최종 모습을 출력하는 문제

제한 사항

- 채우는 방법이 여럿인 경우는 그 중 하나만을 출력

예제

#알튜비튜

예제 입력

```
0 3 5 4 6 9 2 7 8
7 8 2 1 0 5 6 0 9
0 6 0 2 7 8 1 3 5
3 2 1 0 4 6 8 9 7
8 0 4 9 1 3 5 0 6
5 9 6 8 2 0 4 1 3
9 1 7 6 5 2 0 8 0
6 0 3 7 0 1 9 5 2
2 5 8 3 9 4 7 6 0
```

예제 출력

```
1 3 5 4 6 9 2 7 8
7 8 2 1 3 5 6 4 9
4 6 9 2 7 8 1 3 5
3 2 1 5 4 6 8 9 7
8 7 4 9 1 3 5 2 6
5 9 6 8 2 7 4 1 3
9 1 7 6 5 2 3 8 4
6 4 3 7 8 1 9 5 2
2 5 8 3 9 4 7 6 1
```

스도쿠

#알튜비튜

접근

- N-Queen 문제와 유사
- 재귀함수를 설계해보자
  → 모든 빈칸을 차례로 채워나감
  → 빈칸에 1~9까지 다 넣어봄
  → 가로, 세로, 정사각형에 중복되는 수가 있는 경우 가지치기

O(N) 가지치기 방법

#알튜비튜

접근

- 가로 9칸, 세로 9칸, 정사각형 9칸 밖에 안되니까..
  반복문으로 중복 체크
- O(N)

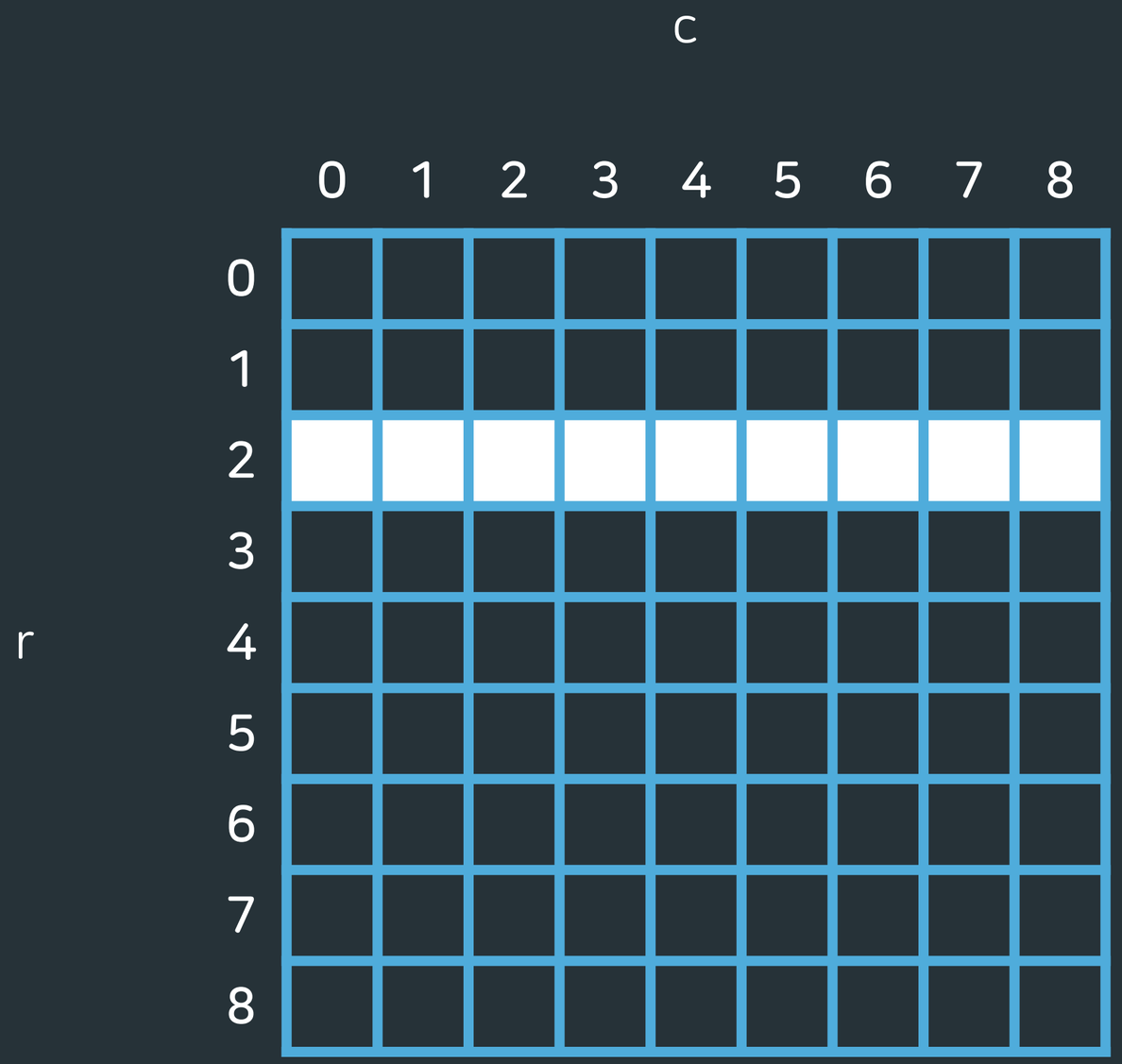
O(1) 가지치기 방법

#알튜비튜

접근

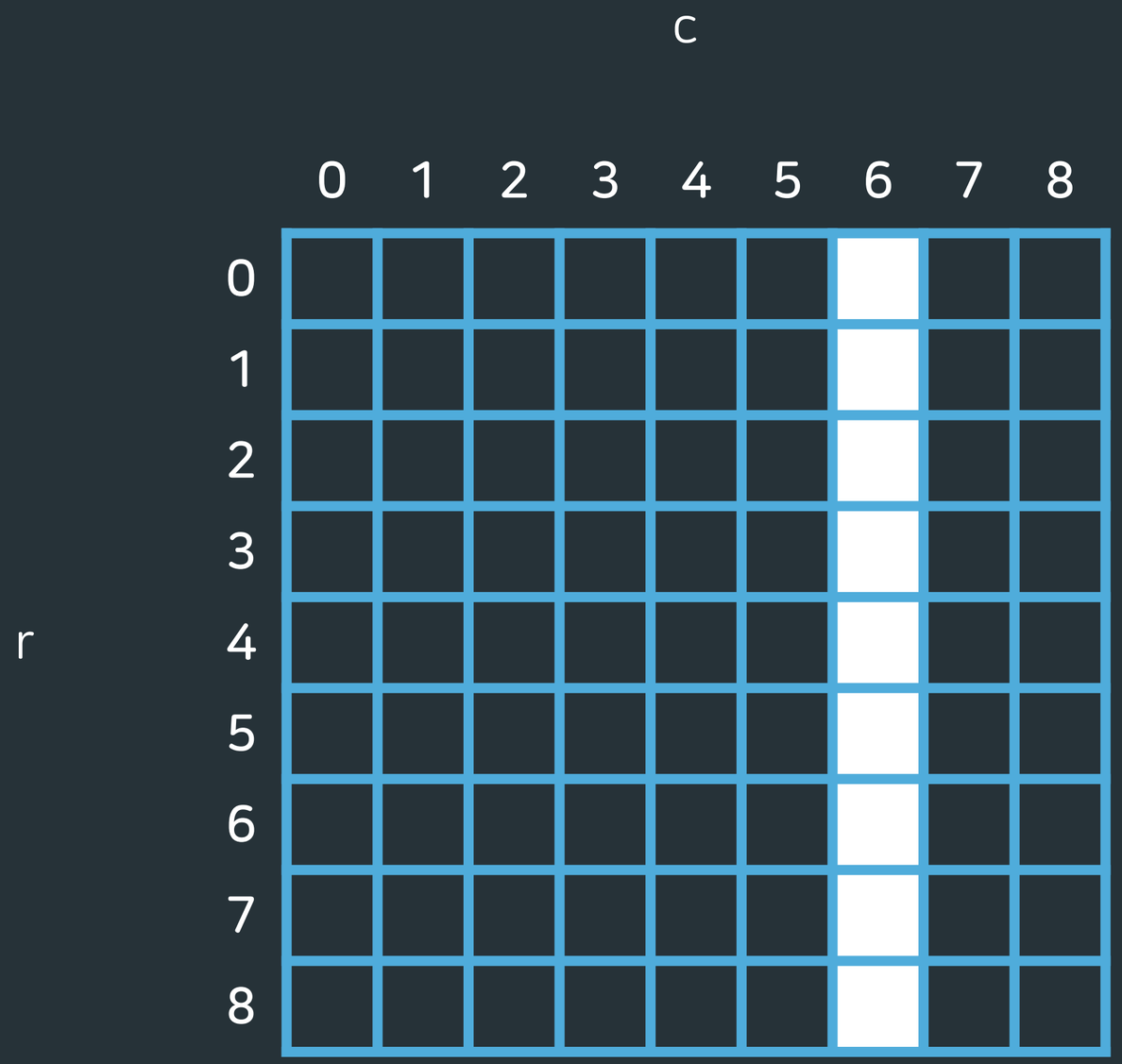
- N-Queen에서 했던 것처럼
  가로, 세로, 정사각형 내 특정 숫자 존재 여부를 bool 배열로 관리
- O(1)
- 단, 퀸처럼 한 가지만 있는 것이 아니라 1~9까지 9종류 있으므로
  2차원 bool 배열로 관리해야 함

가로



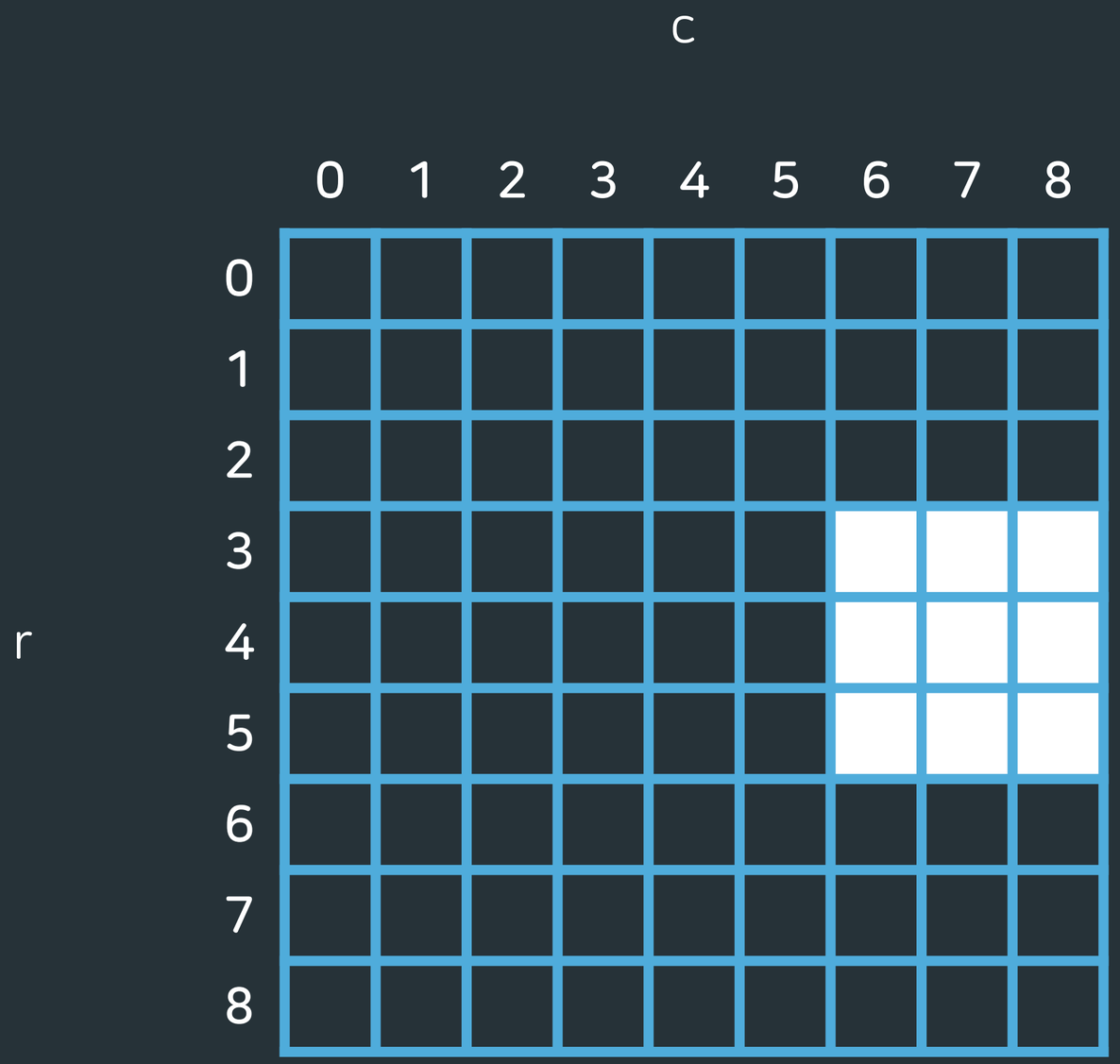
- is_in_row[r][n]
  : r번 행에 n이 존재하는지 여부

세로



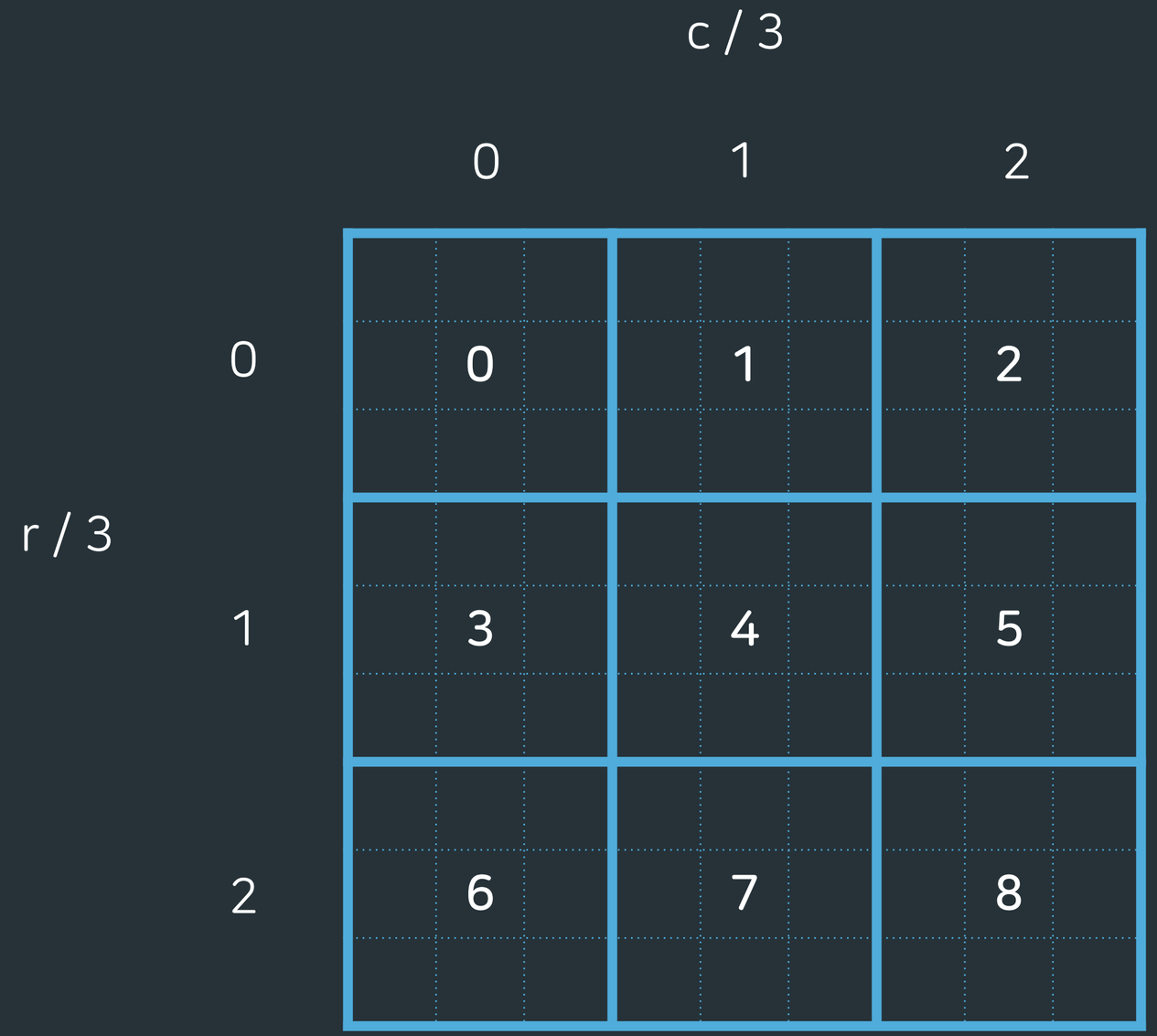
- is_in_col[c][n]
  : c번 열에 n이 존재하는지 여부

정사각형



- is_in_square[s][n]
  : s번 정사각형에
  n이 존재하는지 여부

정사각형 넘버링



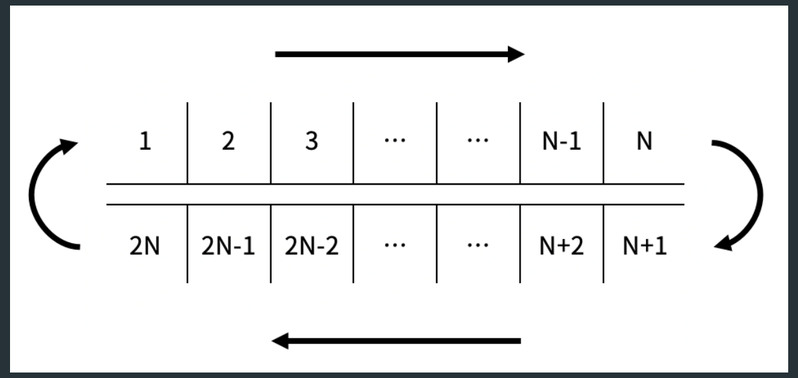
- square(r, c)
  = (r / 3) * 3 + (c / 3)

#알튜비튜

/<> 20055번: 컨베이어 벨트 위의 로봇 - Gold 5

문제

- 벨트의 1번 칸은 "올리는 위치", n번 칸은 "내리는 위치"
- 로봇은 1번 칸에서만 올릴 수 있고, n번 칸에서만 내릴 수 있다.
- 로봇을 올리거나, 로봇이 어떤 칸으로 이동 시 칸의 내구도 -1



로봇이 움직이는 과정

1. 벨트가 각 칸 위의 로봇과 함께 한 칸 회전
2. 가장 먼저 벨트에 올라간 로봇부터, 벨트 회전 방향으로 한 칸 이동할 수 있다면 이동
   (이동 가능: 이동하려는 칸에 로봇이 없고, 그 칸의 내구도가 1 이상이어야 함)
3. 올리는 위치에 있는 칸의 내구도가 0이 아니면 로봇 올림
4. 내구도가 0인 칸의 개수가 k개 이상이라면 과정 종료. 그렇지 않다면 1로 돌아감

-> 1 ~ 3까지가 1단계

문제 해결 방법

#알튜비튜

- 회전과 관련이 깊은 자료구조를 사용하자 -> deque
- 벨트의 한 칸에 대해 "내구도"와 "로봇의 존재여부"를 관리해야 한다 -> struct를 사용하자
- 칸 번호가 1 ~ 2n으로 주어졌지만, 0번부터 시작하는 것으로 하자

  (0번 칸이 로봇을 올리는 칸, n-1번 칸이 로봇을 내리는 칸)

- 문제에서 주어진 과정을 따라가며 구현을 하자
  1. 벨트 회전
  2. 로봇 이동
  3. 로봇 추가

로봇이 움직이는 과정

1. 벨트가 각 칸 위의 로봇과 함께 한 칸 회전
2. 가장 먼저 벨트에 올라간 로봇부터, 벨트 회전 방향으로 한 칸 이동할 수 있다면 이동
   (이동가능: 이동하려는 칸에 로봇이 없고, 그 칸의 내구도가 1 이상이어야 함)
3. 올리는 위치에 있는 칸의 내구도가 0이 아니면 올리는 위치에 로봇 올림
4. 내구도가 0인 칸의 개수가 k개 이상이라면 과정 종료. 그렇지 않다면 1로 돌아감

-> 1 ~ 3까지가 1단계

자세히 봅시다

#알튜비튜

- 벨트의 한 칸에 대해 "내구도"와 "로봇의 존재여부"를 관리해야 한다 -> struct를 사용하자

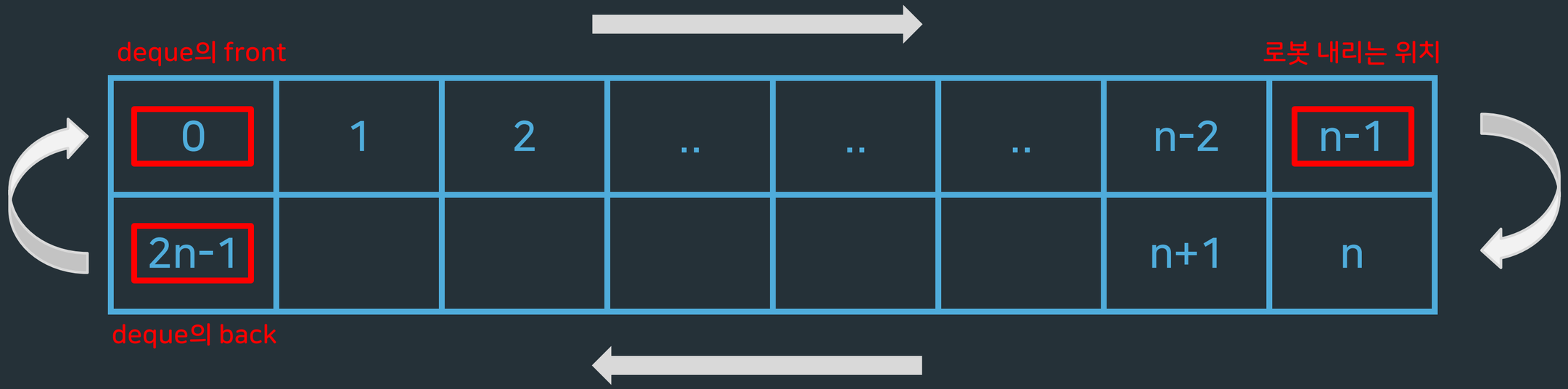
```
struct info {
    int power;  // 내구도
    bool is_on;  // 로봇 존재 여부
};
```

- 회전과 관련이 깊은 자료구조를 사용하자 -> deque

```
deque<info> belt(2 * n); // 컨베이어 벨트의 내구도와 로봇 존재 여부 저장
```

1. 벨트 회전

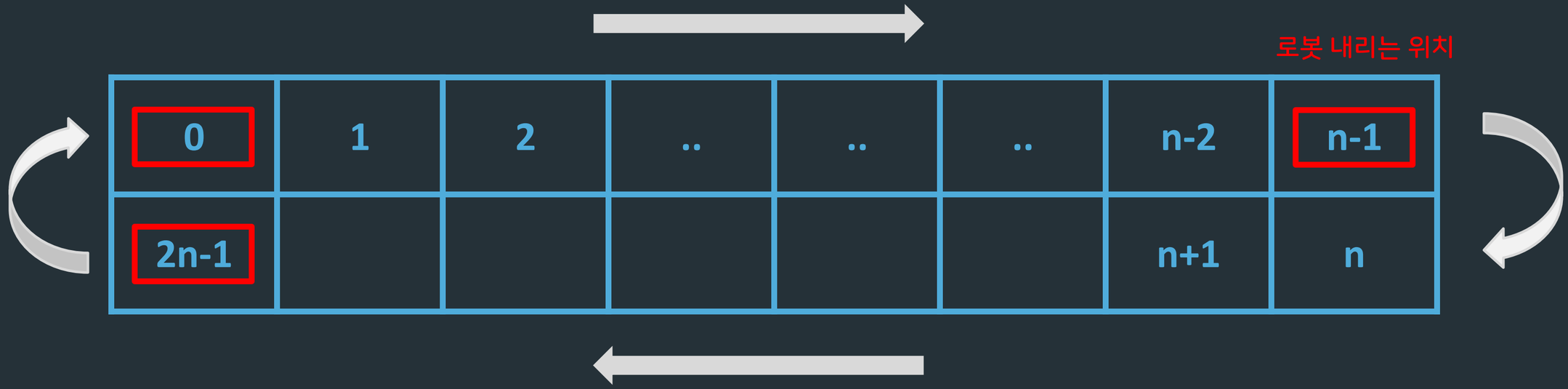
- 벨트 한 칸을 회전하는 함수 rotateBelt()



- 가장 마지막 원소를 처음에 가져다 넣으면 회전 성공
- 내리는 위치에서 로봇을 "반드시" 내려야 함

2. 로봇 이동

- 로봇을 이동시킬 수 있으면 한 칸 이동시키는 함수 moveRobot()
  - 현재 칸에 로봇이 존재하고
  - 다음 칸에 로봇이 없으며
  - 다음 칸의 내구도가 남아있어야 **"이동 가능"**

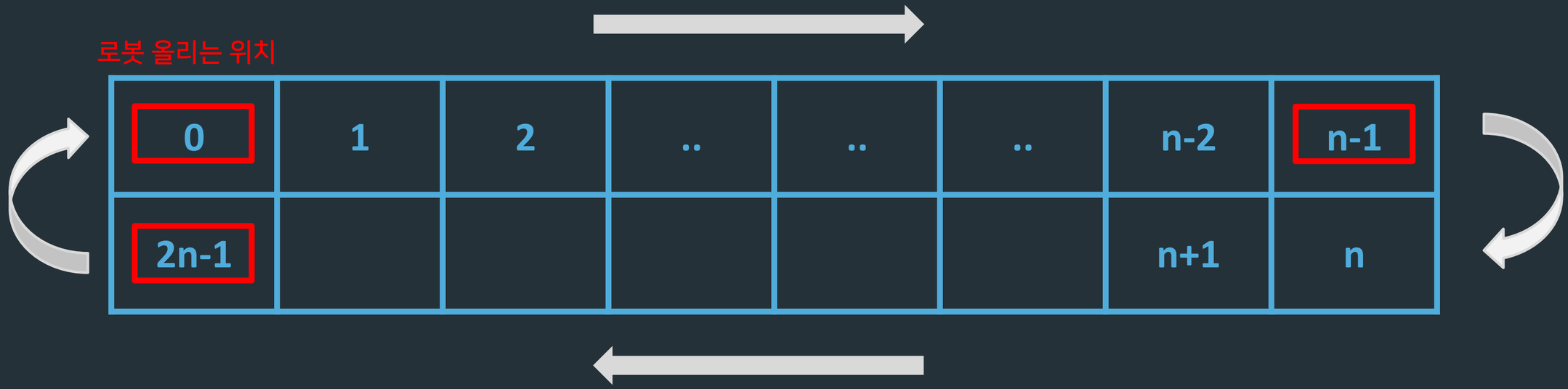


n-1 칸에서 반드시 로봇이 내리기 때문에, 로봇 이동은 0 ~ n-2 칸만 따지면 된다.

로봇을 이동시켰으면 벨트에 로봇 존재여부를 표시하고, 내구도를 감소시켜야 한다.

3. 로봇 추가

- 올리는 칸에 로봇을 올릴 수 있으면 올리는 함수 putRobot()
  - 올리는 칸에 로봇이 존재하지 않으며,
  - 올리는 칸의 내구도가 남아있으면 **"로봇 추가 가능"**



로봇을 올리고 나면, 벨트에 로봇을 추가했음을 표시하고, 내구도를 감소시킬 것

마무리

#알튜비튜

추가로 풀어보면 좋은 문제!

- 15657번 : N과 M (8) – Silver 3
- 10971번 : 외판원 순회 2 – Silver 2
- 14889번: 스타트와 링크 - Silver 2
- 13023번: ABCDE - Gold 5
- 1759번 : 암호 만들기 – Gold 5
- 15811번 : 복면산?! – Gold 4