코딩 테스트 대비 핵심 알고리즘

핵심 유형 문제풀이

핵심 유형 문제풀이 | 다양한 문제를 접하며 코딩 테스트에 익숙해지기

강사 나동빈

코딩 테스트 대비
핵심 알고리즘

핵심 유형 문제풀이

혼자 힘으로 풀어보기

문제 제목: 컨베이어 벨트 위의 로봇

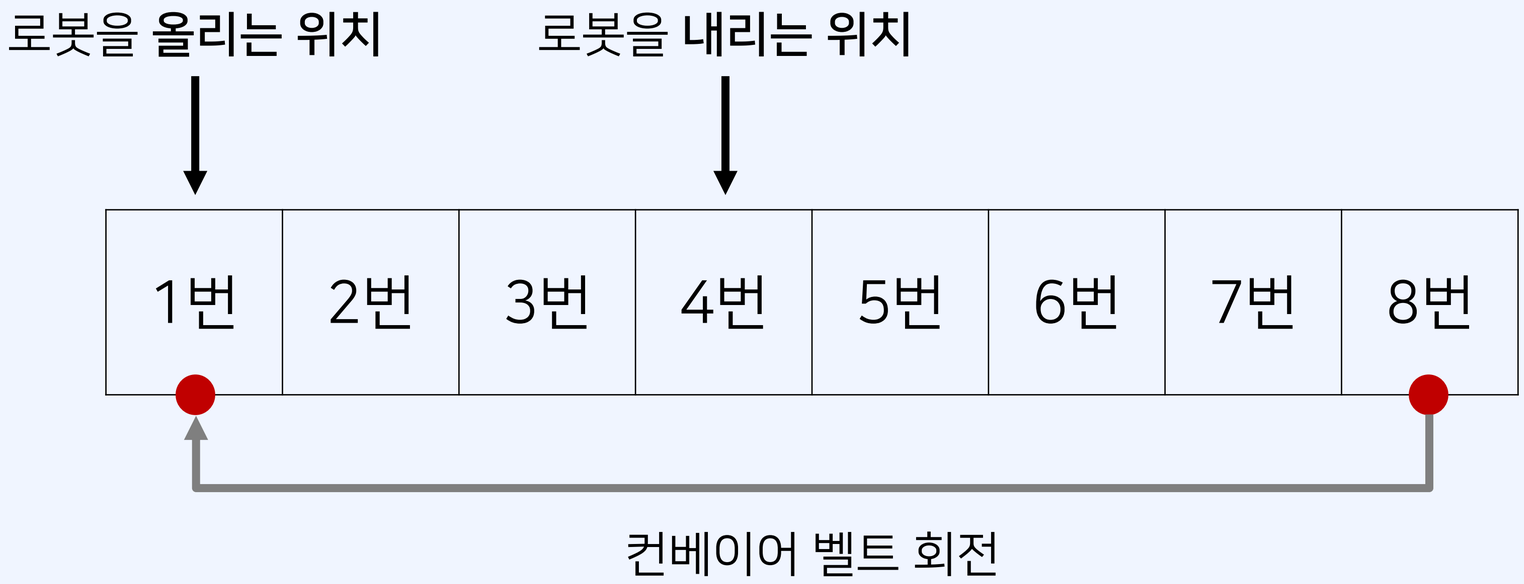
문제 난이도: ★★★☆☆

문제 유형: 시뮬레이션

추천 풀이 시간: 60분

문제 해결 아이디어

- 문제의 요구사항 그대로 구현하면 되는 **시뮬레이션** 유형의 문제다.
- 컨베이어 벨트는 원형으로 구성되므로, 총길이를 $2N$으로 간주할 수 있다.
- $N = 4$일 때의 예시는 다음과 같다.



컨베이어 벨트 위의 로봇: https://www.acmicpc.net/problem/20055

문제 해결 아이디어

단계별 수행

1. 벨트가 각 칸 위에 있는 로봇과 함께 **한 칸 회전**한다.
2. 가장 먼저 벨트에 올라간 로봇부터, 벨트가 회전하는 방향으로 한 칸 이동할 수 있다면 이동한다.
   로봇이 이동하기 위해서는 이동하려는 칸에 로봇이 없으며, 내구도가 1 이상 남아 있어야 한다.
3. 올리는 위치에 있는 칸의 내구도가 0이 아니면 **올리는 위치에 로봇을 올린다.**

내리는 위치에 도달한 로봇은 그 즉시 내린다.

로봇이 올라가거나, 어떠한 칸으로 이동할 때 그 칸의 내구도는 1만큼 감소한다.

초기 상태

10	1	10	6	3	4	8	2

로봇을 **올리는 위치** (1번 칸)　　로봇을 **내리는 위치** (4번 칸)

로봇

1단계

1	10	1	10	6	3	4	8

문제 해결 아이디어

컨베이어 벨트 위의 로봇: https://www.acmicpc.net/problem/20055

문제 해결 아이디어

로봇을 올리는 위치 ↓ 로봇을 내리는 위치 ↓

4단계	2	3	6	0	9	1	10	6
	로봇		로봇					
5단계	5	2	2	6	0	9	1	10
	로봇		로봇					
6단계	9	5	1	2	6	0	9	1
	로봇		로봇					
7단계	0	9	4	1	2	6	0	9
	로봇		로봇					

컨베이어 벨트 위의 로봇: https://www.acmicpc.net/problem/20055

소스 코드


```python
import sys
# 빠른 입력 함수 사용
input = sys.stdin.readline
# 원형 큐를 위해 덱(queue) 사용
from collections import deque

n, k = map(int, input().split())
belt = deque(list(map(int, input().split())))
robot = deque([False] * n)
result = 1 # 1단계부터 시작

while True:
    # 1. 벨트가 로봇과 함께 한 칸 회전
    belt.rotate()
    robot.rotate()
    robot[n - 1] = False # 내리는 위치에서 로봇 내리기
    # 2. 가장 먼저 벨트에 올라간 로봇부터,
    # 벨트가 회전하는 방향으로 한 칸 이동할 수 있다면 이동
    for i in range(n - 2, -1, -1):
        if robot[i] and not robot[i + 1] and belt[i + 1] >= 1:
            robot[i] = False
            robot[i + 1] = True
            belt[i + 1] -= 1
    robot[n - 1] = False # 내리는 위치에서 로봇 내리기
    # 3. 올리는 위치에 있는 칸의 내구도가 0이 아니면,
    # 올리는 위치에 로봇을 올리기
    if belt[0] >= 1:
        robot[0] = True
        belt[0] -= 1
    # 4. 내구도 0이 K개 이상이라면 종료
    if belt.count(0) >= k:
        break
    result += 1
print(result)
```

혼자 힘으로 풀어보기

문제 제목: 마법사 상어와 토네이도

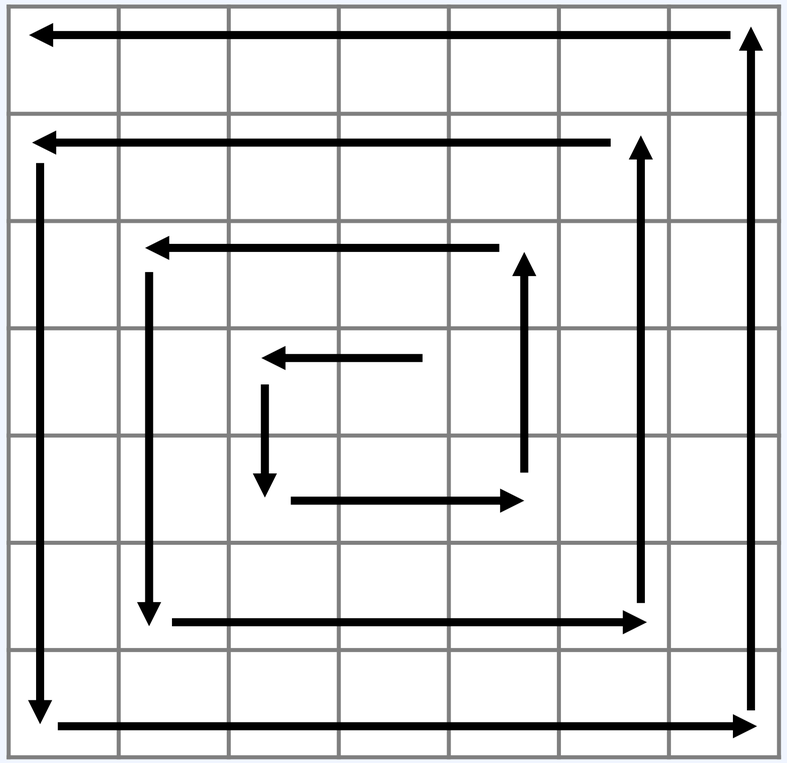
문제 난이도: ★★★☆☆

문제 유형: 시뮬레이션

추천 풀이 시간: 70분

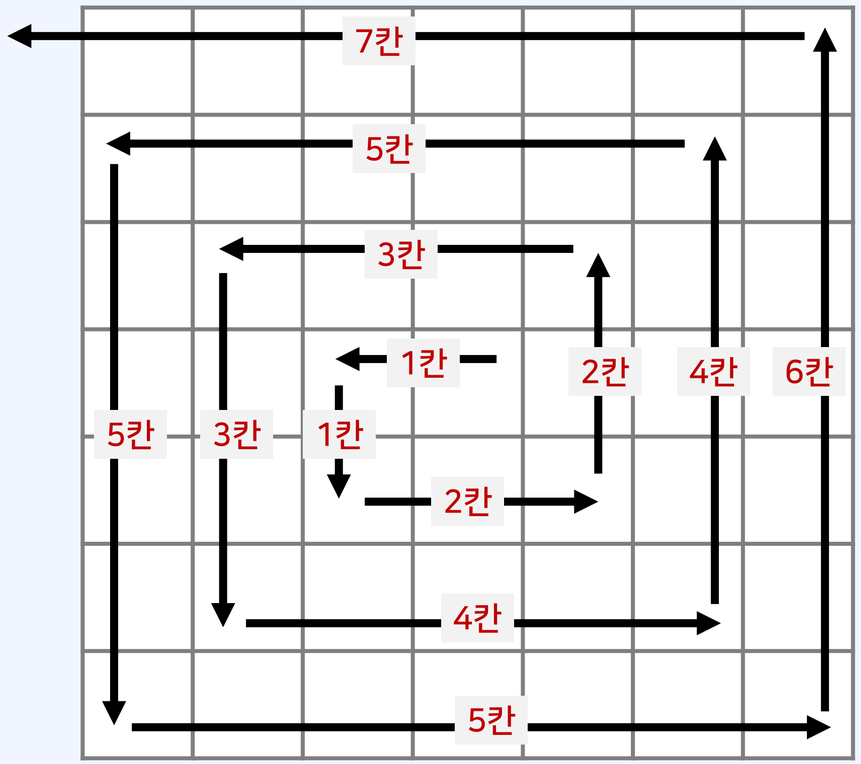
문제 해결 아이디어

- 문제의 요구사항 그대로 구현하면 되는 **시뮬레이션** 유형의 문제다.
- $N = 7$ 크기의 격자가 있을 때, **가운데 칸부터 토네이도의 이동이 시작**된다.



문제 해결 아이디어

- **토네이도가 이동할 때의 규칙**은 다음과 같다.
- 서 1칸, 남 1칸 → 동 2칸, 북 2칸 → 서 3칸, 남 3칸 → 동 4칸, 북 4칸, …




```
direction = 0 # 방향(처음엔 서쪽)
turned = 0 # 회전한 횟수
moved = 0 # 현재 방향으로 이동한 수
target = 1 # 이동할 칸 수

def move():
    global x, y, direction, turned, moved, target
    if moved == target: # 충분히 이동했다면 회전 수행
        moved = 0
        turned += 1
        direction = (direction + 1) % 4
    if turned == 2: # 2번 회전했다면, 이동할 칸 수 증가
        turned = 0
        target += 1
    # 다음 위치로 이동
    x = x + dx[direction]
    y = y + dy[direction]
    moved += 1
```

마법사 상어와 토네이도: https://www.acmicpc.net/problem/20057

문제 해결 아이디어

- **이동 방향에 따른 모래의 비율**은 다음과 같다.

왼쪽(Left)

		2%		
	10%	7%	1%	
5%	α	y ←	x	
	10%	7%	1%	
		2%		

아래쪽(Down)

	1%	x	1%	
2%	7%	y ↓	7%	2%
	10%	α	10%	
		5%		

오른쪽(Right)

		2%			
	1%	7%	10%		
	x	→ y	α	5%	.....
	1%	7%	10%		
		2%			

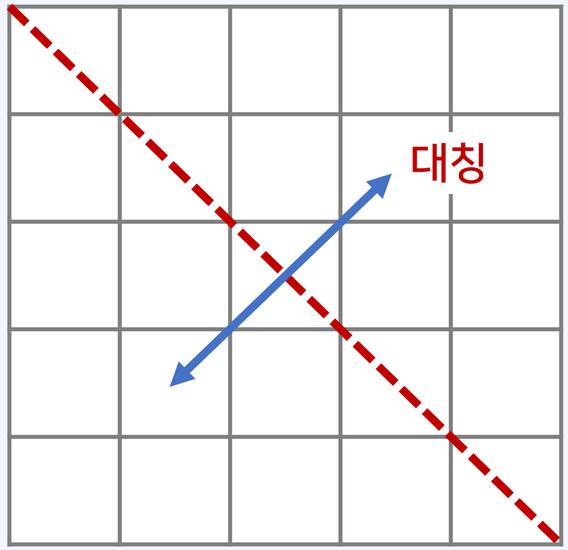
문제 해결 아이디어

- $N \times N$ 행렬의 x와 y 좌표를 서로 바꾸는 경우, **전치 행렬**을 얻을 수 있다.

```
up = [(y, x, val) for x, y, val in left]
```

왼쪽(Left)

		2%		
	10%	7%	1%	
5%	α	y ←	x	
	10%	7%	1%	
		2%		



위쪽(Up)

		5%		
	10%	α	10%	
2%	7%	y	7%	2%
	1%	x	1%	

문제 해결 아이디어

- 결과적으로 **이동 방향에 따른 모래의 비율**을 코드로 표현하면 다음과 같다.

왼쪽(Left)

		2%		
	10%	7%	1%	
5%	α	y ←	x	
	10%	7%	1%	
		2%		

```
# 방향별 모래 비율
left = [
  (-2, 0, 0.02), (-1, -1, 0.10), (-1, 0, 0.07),
  (-1, 1, 0.01), (0, -2, 0.05), (1, -1, 0.10),
  (1, 0, 0.07), (1, 1, 0.01), (2, 0, 0.02)
]
down = [(-y, x, val) for x, y, val in left]
right = [(x, -y, val) for x, y, val in left]
up = [(y, x, val) for x, y, val in left]
ratio = [left, down, right, up]
```

마법사 상어와 토네이도: https://www.acmicpc.net/problem/20057

소스 코드 1)


```python
import sys
# 빠른 입력 함수 사용
input = sys.stdin.readline

# 서, 남, 동, 북 방향 정보
dx = [0, 1, 0, -1]
dy = [-1, 0, 1, 0]
# 방향별 모래 비율
left = [
  (-2, 0, 0.02), (-1, -1, 0.10), (-1, 0, 0.07),
  (-1, 1, 0.01), (0, -2, 0.05), (1, -1, 0.10),
  (1, 0, 0.07), (1, 1, 0.01), (2, 0, 0.02)
]
down = [(-y, x, val) for x, y, val in left]
right = [(x, -y, val) for x, y, val in left]
up = [(y, x, val) for x, y, val in left]
ratio = [left, down, right, up]
```

```python
def move():
    global x, y, direction, turned, moved, target
    if moved == target: # 충분히 이동했다면 회전 수행
        moved = 0
        turned += 1
        direction = (direction + 1) % 4
    if turned == 2: # 2번 회전했다면, 이동할 칸 수 증가
        turned = 0
        target += 1
    # 다음 위치로 이동
    x = x + dx[direction]
    y = y + dy[direction]
    moved += 1
```

마법사 상어와 토네이도: https://www.acmicpc.net/problem/20057

소스 코드 2)


```python
n = int(input()) # 맵의 크기
arr = []
for i in range(n):
    arr.append(list(map(int, input().split())))
x = n // 2
y = n // 2
direction = 0 # 방향(처음엔 서쪽)
turned = 0 # 회전한 횟수
moved = 0 # 현재 방향으로 이동한 수
target = 1 # 이동할 칸 수
result = 0 # 맵을 벗어나는 모래의 양
cnt = 1 # 총 이동 횟수
```

```python
while cnt < n * n:
    move() # 한 칸 이동
    remain = arr[x][y] # 남은 모래의 양
    for i in range(9): # 각 9개의 위치로 모래 옮기기
        nx, ny, percentage = ratio[direction][i]
        nx += x
        ny += y
        current = int(arr[x][y] * percentage) # 옮길 모래 양
        # 맵을 벗어나는 경우
        if nx < 0 or nx >= n or ny < 0 or ny >= n:
            result += current
        else:
            arr[nx][ny] += current
        remain -= current
    # 알파(alpha) 값 처리하기(남은 모래 옮기기)
    nx = x + dx[direction]
    ny = y + dy[direction]
    # 맵을 벗어나는 경우
    if nx < 0 or nx >= n or ny < 0 or ny >= n:
        result += remain
    else:
        arr[nx][ny] += remain
    arr[x][y] = 0
    cnt += 1
print(result)
```