

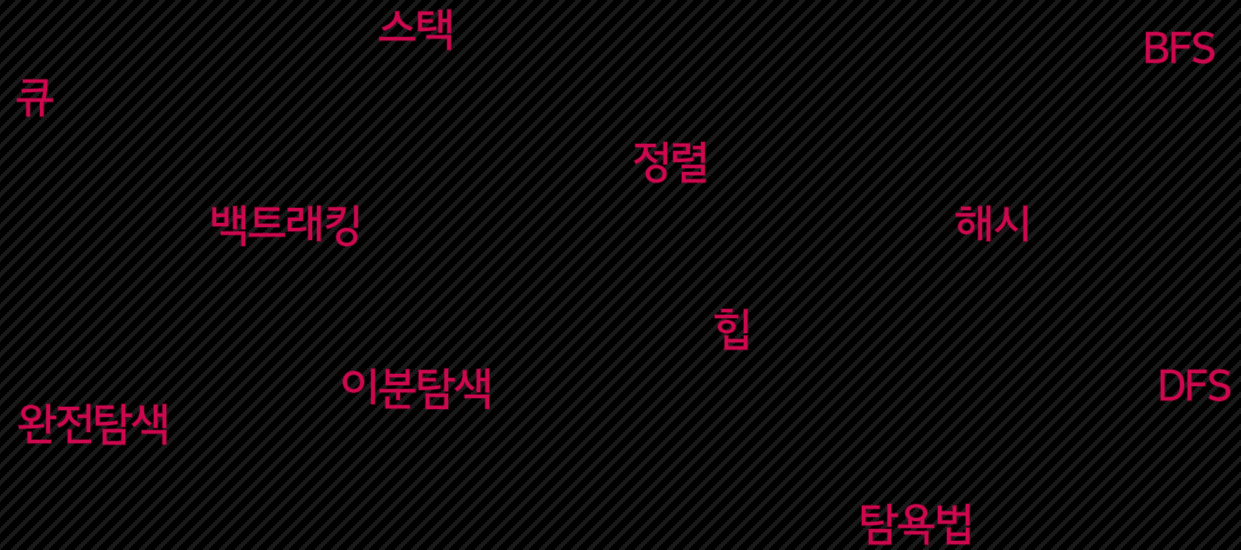


# 알고리즘 특강 구현력 트레이닝

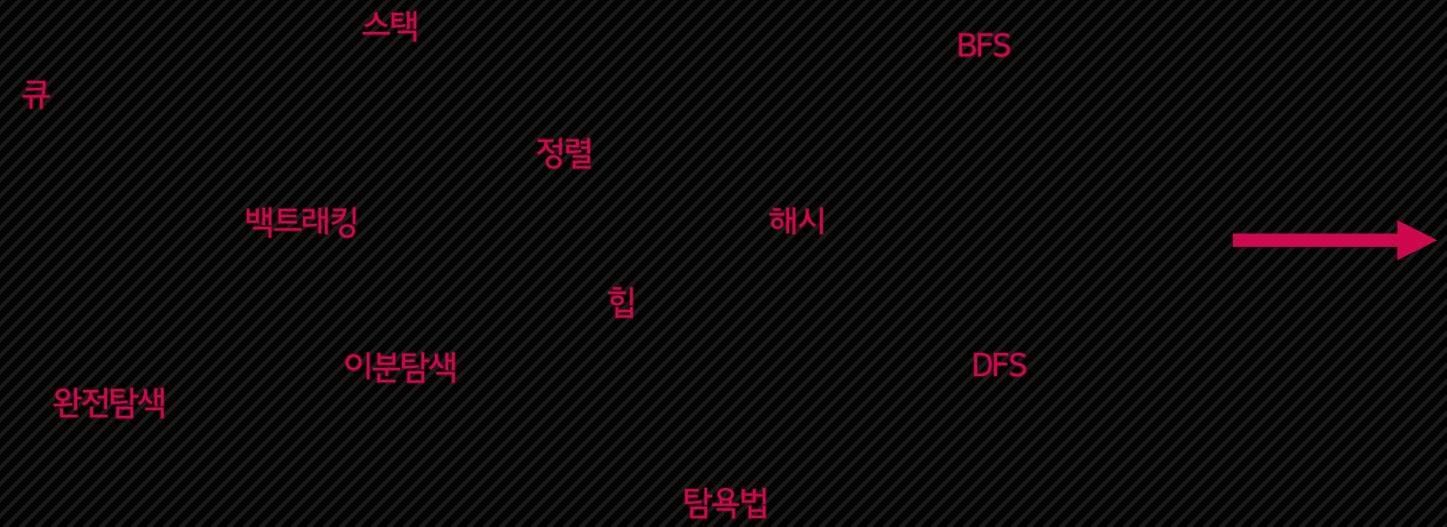
---

코딩테스트의 핵심파트, 구현입니다.  
꼼꼼함과 분석이 가장 중요합니다.

## 지금까지 배운 것들...



구현?



구현

## 코딩테스트의 측면에서

### 삼성 공채 / SW 역량 테스트 A형

- 3시간 동안 2문항 출제. 2문항 모두 구현 문제
- **히든 테스트 케이스** 존재. 주어진 테스트 케이스 맞췄다고 합격인 것이 아님!

### 카카오 블라인드 채용

- 구현 문제가 아니더라도 전체적으로 **기본적인 문자열 처리 및 설계 능력**이 필요함.
- 구현 문제의 **난이도가 높은 편**. (정답률 5~10% 미만!)

### 네이버, 라인, 쿠팡...

- 짧은 제한 시간 내에 문제를 해결해야 함.
- 제공되는 **테스트 케이스 1개**. 이외 모든 케이스는 **전부 히든 케이스!**

## 어떻게 풀까?

### 문제 분석

- 문제에서 요구하는 바를 읽고, 어떤 절차를 통해 문제를 풀어야 하는지 설계.
- 구현 문제는 정보량이 많은 문제가 상당하니 주의 깊게 읽고 설계해야 함!

### 코드 구조 설계

- 설계한 풀이를 실제로 코드로 옮기기 위해 필요한 구현 방식, 변수형 등을 설계함.
- 실수를 줄이기 위해, 내부 구현의 길이를 줄이고 컴포넌트를 많이 설계하는게 좋음!

### 컴포넌트 별 구현 및 테스트

- 설계 내용을 바탕으로 컴포넌트마다 구현 작업을 진행.
- 코드를 다 짰 이후 테스트 하는 것 보다, 각각의 컴포넌트 작업 완료 후 테스트 하는 것을 권장함!



## 문제를 풀어봅시다!

 Gold 5 - 치킨 배달 (#15686, 삼성 SW 역량테스트 A형 유사문항)

 Gold 5 - 인내의 도미노 장인 호석 (#20165)

 Level 2 - 프렌즈4블록 (kakao 2018 Blind Recruitment 1차 코딩테스트)

 Gold 5 - 미세먼지 안녕! (#17144, 삼성 SW 역량테스트 A형 유사문항)

 Gold 4 -  (#17281, 삼성 SW 역량테스트 A형 유사문항)

 Gold 3 - 독특한 계산기 (#19591)

※ 해당 단원은 '문제 분석'의 충분한 연습을 위해 문제 요약에 기술하지 않습니다.

## 치킨 배달

### 문제 분석

크기가  $N \times N$ 인 도시가 있다. 도시는  $1 \times 1$ 크기의 칸으로 나누어져 있다.

도시의 각 칸은 빈 칸, 치킨집, 집 중 하나이다. 도시의 칸은  $(r, c)$ 와 같은 형태로 나타내고,

$r$ 행  $c$ 열 또는 위에서부터  $r$ 번째 칸, 왼쪽에서부터  $c$ 번째 칸을 의미한다.  $r$ 과  $c$ 는 1부터 시작한다.

이 도시에 사는 사람들은 치킨을 매우 좋아한다. 따라서, 사람들은 "치킨 거리"라는 말을 주로 사용한다.

치킨 거리는 집과 가장 가까운 치킨집 사이의 거리이다.

즉, 치킨 거리는 집을 기준으로 정해지며, 각각의 집은 치킨 거리를 가지고 있다.

도시의 치킨 거리는 모든 집의 치킨 거리의 합이다.

문제를 풀기전에 종이와 펜을 꺼내 봅시다!

## 치킨 배달

### 문제 분석

크기가  $N \times N$ 인 도시가 있다. 도시는  $1 \times 1$ 크기의 칸으로 나누어져 있다.

도시의 각 칸은 빈 칸, 치킨집, 집 중 하나이다. 도시의 칸은  $(r, c)$ 와 같은 형태로 나타내고,  $r$ 행  $c$ 열 또는 위에서부터  $r$ 번째 칸, 왼쪽에서부터  $c$ 번째 칸을 의미한다.  $r$ 과  $c$ 는 1부터 시작한다.

→ 그래프 문제 같네? 그런데  $r$ 과  $c$ 가 0이 아니라 1부터 시작한다고 하네... 일단 체크하자.



# 치킨 배달

## 문제 분석

크기가  $N \times N$ 인 도시가 있다. 도시는  $1 \times 1$ 크기의 칸으로 나누어져 있다.

도시의 각 칸은 빈 칸, 치킨집, 집 중 하나이다. 도시의 칸은  $(r, c)$ 와 같은 형태로 나타내고,  $r$ 행  $c$ 열 또는 위에서부터  $r$ 번째 칸, 왼쪽에서부터  $c$ 번째 칸을 의미한다.  $r$ 과  $c$ 는 1부터 시작한다.

→ 그래프 문제 같네? 그런데  $r$ 과  $c$ 가 0이 아니라 1부터 시작한다고 하네... 일단 체크하자.

이 도시에 사는 사람들은 치킨을 매우 좋아한다. 따라서, 사람들은 "치킨 거리"라는 말을 주로 사용한다.

치킨 거리는 집과 가장 가까운 치킨집 사이의 거리이다.

즉, 치킨 거리는 집을 기준으로 정해지며, 각각의 집은 치킨 거리를 가지고 있다.

도시의 치킨 거리는 모든 집의 치킨 거리의 합이다.

→ 가장 가까운 치킨집 사이의 거리 = 최단거리!  
결국 최단거리의 합을 구하는 문제라고 볼 수 있겠구나!

## 치킨 배달

### 문제 분석

이 도시에 있는 치킨집은 모두 같은 프랜차이즈이다.

프랜차이즈 본사에서는 수익을 증가시키기 위해 **일부 치킨집을 폐업 시키려고** 한다.

오랜 연구 끝에 이 도시에서 가장 수익을 많이 낼 수 있는 치킨집의 개수는 **최대 M개**라는 사실을 알아내었다.

도시에 있는 치킨집 중에서 최대 M개를 고르고, 나머지 치킨집은 모두 **폐업 시켜야** 한다.

어떻게 고르면, 도시의 **치킨 거리가 가장 작게 될지 구하는** 프로그램을 작성하시오.

- M개를 빼고 모두 폐업한다 = 현재 있는 치킨 집 중에 M개를 선택한다!  
M개를 선택해서 최단거리의 합을 구해주고, 그것의 최솟값을 출력하면 될 것 같아!

# 치킨 배달

## 문제 분석

- 그래프 문제 같네? 그런데  $r$ 과  $c$ 가 0이 아니라 1부터 시작한다고 하네... 일단 체크하자.
- 가장 가까운 치킨집 사이의 거리 = 최단거리!  
결국 최단거리의 합을 구하는 문제라고 볼 수 있겠구나!
- $M$ 개를 빼고 모두 폐업한다 = 현재 있는 치킨 집 중에  $M$ 개를 선택한다!  
 $M$ 개를 선택해서 최단거리의 합을 구해주고, 그것의 최솟값을 출력하면 될 것 같아!

# 치킨 배달


## 문제 분석

- 그래프 입력 받기
- 치킨집 중에서 M개를 선택
- 선택한 M개에 대해 최단거리 구하기
- 기존 최단거리의 합과 비교해 작으면 갱신

# 치킨 배달

## 문제 분석

- 그래프 입력 받기
  - 치킨집 선택을 하기 위해선 미리 목록을 뽑아야 할 것 같은데...
- 치킨집 중에서 M개를 선택
  - 선택한 것과 선택하지 않은 건 어떻게 구분하지?
- 선택한 M개에 대해 최단거리 구하기
  - 탐색을 시도하는 곳도 여러 곳이고, 목적지도 여러 곳인데 최단거리를 어떻게 찾지?
- 기존 최단거리의 합과 비교해 작으면 갱신



큰 구조를 먼저 생각하고,  
세부 사항을 내용 사이에 넣어서 고민해보자!

# 치킨 배달

## 코드 구조 및 설계

- 그래프 입력 받기
  - 치킨집 선택을 하기 위해선 미리 목록을 뽑아야 할 것 같은데...
- 치킨집 중에서 M개를 선택
  - 선택한 것과 선택하지 않은 건 어떻게 구분하지?
- 선택한 M개에 대해 최단거리 구하기
  - 탐색을 시도하는 곳도 여러 곳이고, 목적지도 여러 곳인데 최단거리를 어떻게 찾지?
- 기존 최단거리의 합과 비교해 작으면 갱신



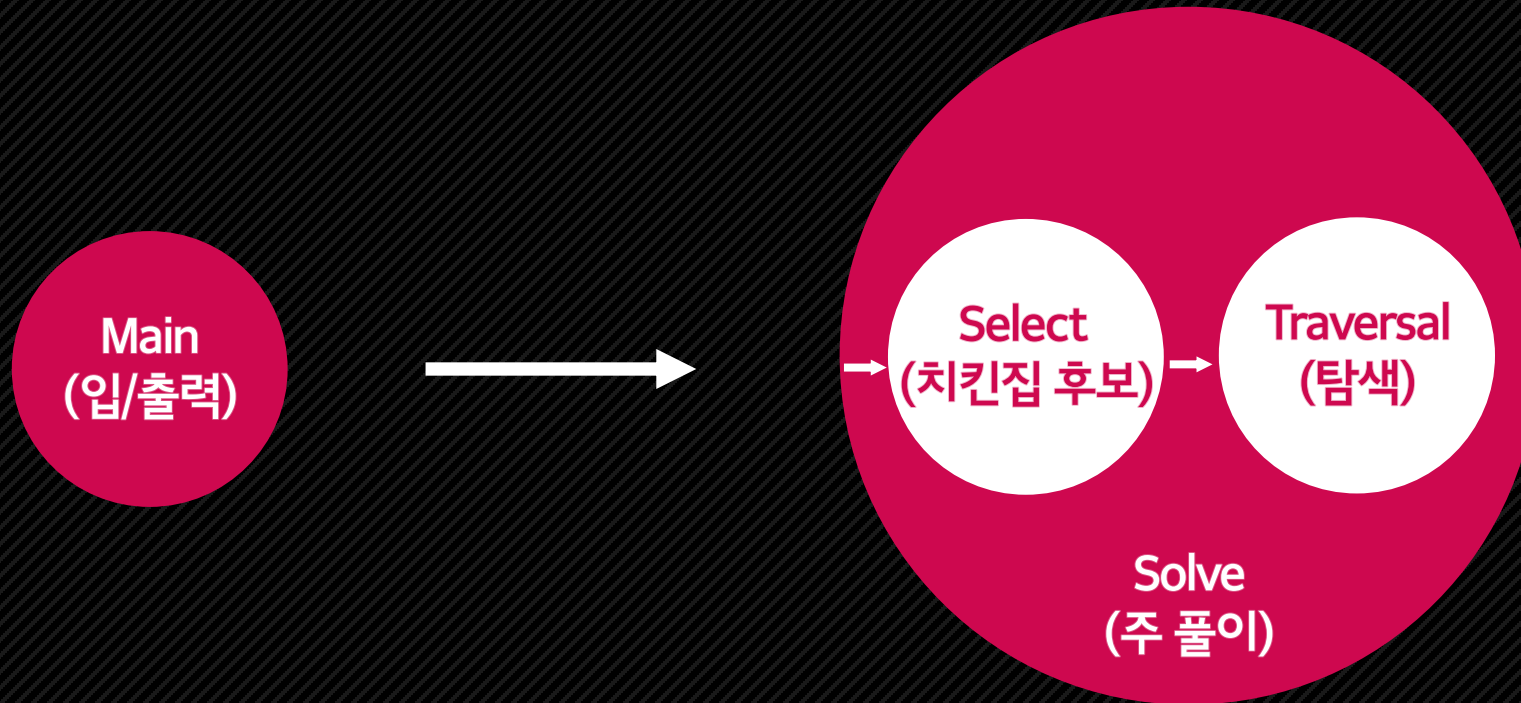
데이터 입력

데이터 선택 후 탐색 전 처리

탐색

## 치킨 배달

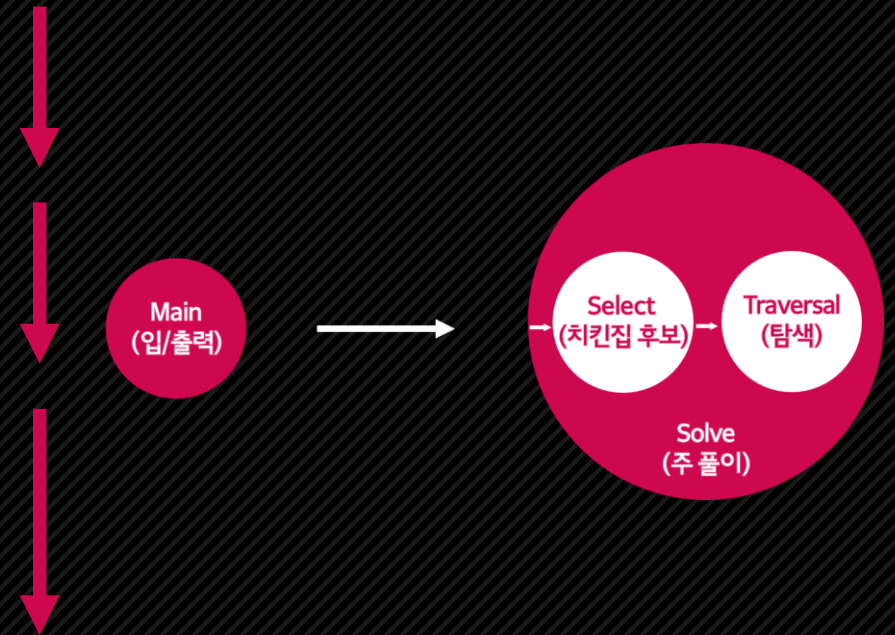
## 코드 구조 및 설계



# 치킨 배달

## 코드 구조 및 설계

- 그래프 입력 받기
  - 치킨집 선택을 하기 위해선 미리 목록을 뽑아야 할 것 같은데...
- 치킨집 중에서 M개를 선택
  - 선택한 것과 선택하지 않은 건 어떻게 구분하지?
- 선택한 M개에 대해 최단거리 구하기
  - 탐색을 시도하는 곳도 여러 곳이고, 목적지도 여러 곳인데 최단거리를 어떻게 찾지?
- 기존 최단거리의 합과 비교해 작으면 갱신





# 치킨 배달

## 코드 구조 및 설계

```
def getMinimumPath():  
    result = 0  
    # 그래프 탐색  
    return result  
  
def select():  
    # M개를 선택할 수 있는 방법은 뭐가 있을까?  
    getMinimumPath()
```

```
def solve():  
    select()  
  
if __name__ == "__main__":  
    # 데이터 입력 받기  
    solve()  
    print(answer)
```

## 인내의 도미노 장인 호석



**잠깐!**

뒤로 넘어가기 전,  
딱 **5분만** 직접 문제를 읽고 분석해봅시다.

## 인내의 도미노 장인 호석

- $N$ 행  $M$ 열의 2차원 격자 모양의 게임판의 각 격자에 도미노를 세운다. 각 도미노는 1 이상 5 이하의 높이를 가진다.
- 매 라운드는 공격수가 먼저 공격하고, 수비수는 공격이 끝난 뒤에 수비를 한다.

특정 행동을 시뮬레이션 하는 문제는 직접 예제를 따라가보는 것이 문제를 이해하는데 좋아!  
설마 아직도 앞에 펜과 종이 없지는 아니죠?

## 인내의 도미노 장인 호석

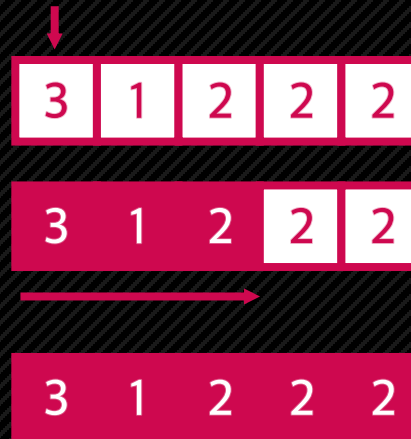
- 공격수는 특정 격자에 놓인 도미노를 동, 서, 남, 북 중 원하는 방향으로 넘어뜨린다.
- 길이가 K인 도미노가 넘어진다면, 그 방향으로 K-1 개의 도미노들 중 넘어지지 않은 것들이 같은 방향으로 연달아 넘어진다.
- 이때, 당연히 도미노의 특성상, 연쇄적으로 도미노가 넘어질 수 있다.
- 이미 넘어진 도미노가 있는 칸을 공격한 경우에는 아무 일이 일어나지 않는다.



## 인내의 도미노 장인 호석

### 공격수

- 공격수는 특정 격자에 놓인 도미노를 동, 서, 남, 북 중 원하는 방향으로 넘어뜨린다.
- 길이가 K인 도미노가 넘어진다면, 그 방향으로 K-1 개의 도미노들 중 넘어지지 않은 것들이 같은 방향으로 연달아 넘어진다.
- 이때, 당연히 도미노의 특성상, 연쇄적으로 도미노가 넘어질 수 있다.
- 이미 넘어진 도미노가 있는 칸을 공격한 경우에는 아무 일이 일어나지 않는다.



## 인내의 도미노 장인 호석

### 수비수

- 넘어져 있는 도미노들 중에 원하는 것 하나를 다시 세울 수 있다.
- 넘어지지 않은 도미노를 세우려고 하면 아무 일이 일어나지 않는다.

3 1 2 2 2

# 인내의 도미노 장인 호석

## 문제 분석

- 데이터 입력 받기
- 공격수의 턴
- 수비수의 턴
- 최종 결과 출력

# 인내의 도미노 장인 호석

## 코드 구조 및 설계

- 데이터 입력 받기

- 공격수의 턴

→ 현재 세워져 있는지, 아스피를 구분하는 테이블과 길이 테이블을 분리하자.

→ 그 이후에는 하나씩 쓰러뜨릴 때 마다 이동횟수를 차감하는 건 어떨까?

- 수비수의 턴

→ 결과적으로는 선택해서 값을 1로 바꿔주면 될 것 같은데...

- 최종 결과 출력



# 인내의 도미노 장인 호석

## 코드 구조 및 설계

- 데이터 입력 받기

- 공격수의 턴

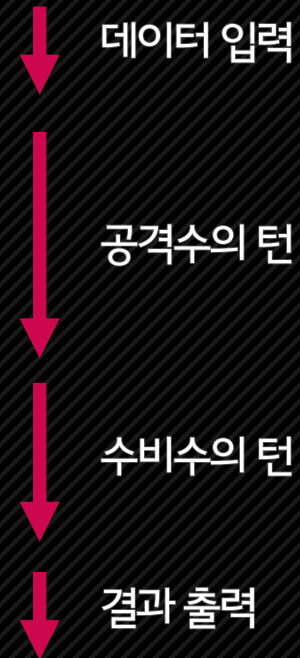
→ 현재 세워져 있는지, 아스피드를 구분하는 테이블과 길이 테이블을 분리하자.

→ 그 이후에는 하나씩 쓰러뜨릴 때 마다 이동횟수를 차감하는 건 어떨까?

- 수비수의 턴

→ 결과적으로는 선택해서 값을 1로 바꿔주면 될 것 같은데...

- 최종 결과 출력



# 인내의 도미노 장인 호석

## 코드 구조 및 설계

```
def offense(x, y, direc):  
    # Some Code...
```

```
def defense(x, y):  
    # Some Code...
```

```
def solve():  
    for i in range(TC):  
        # 공격 정보 데이터 입력  
        offense(x, y, direc)  
        # 수비 정보 데이터 입력  
        defense(x, y)
```

```
if __name__ == "__main__":  
    # 데이터 입력  
    solve()  
    # 데이터 출력
```



도전!

지금까지 활용한 문제 분석 방법을 토대로  
직접 문제를 풀어봅시다!

미세먼지 안녕!



**잠깐!**

뒤로 넘어가기 전,  
딱 **5분만** 직접 문제를 읽고 분석해봅시다.

## 미세먼지 안녕!

- 1초 동안 아래 적힌 일이 순서대로 일어난다.
- 미세먼지가 확산된다. 확산은 미세먼지가 있는 모든 칸에서 동시에 일어난다.
  - (r, c)에 있는 미세먼지는 **인접한 네 방향으로 확산된다.**
  - 인접한 방향에 **공기청정기**가 있거나, 칸이 없으면 그 방향으로 **확산이 일어나지 않는다.**
  - 확산되는 양은  $A_{r,c}/5$ 이고 **소수점은 버린다.**
  - (r, c)에 남은 미세먼지의 양은  $A_{r,c} - (A_{r,c}/5) \times (\text{확산된 방향의 개수})$  이다.

→ 미세먼지 확산을 구현할 때 유의할 점이 있을까?

## 미세먼지 안녕!

- 1초 동안 아래 적힌 일이 순서대로 일어난다.
- 미세먼지가 확산된다. 확산은 미세먼지가 있는 모든 칸에서 동시에 일어난다.
  - (r, c)에 있는 미세먼지는 **인접한 네 방향으로 확산된다.**
  - 인접한 방향에 **공기청정기**가 있거나, 칸이 없으면 그 방향으로 **확산이 일어나지 않는다.**
  - 확산되는 양은  $A_{r,c}/5$ 이고 **소수점은 버린다.**
  - (r, c)에 남은 미세먼지의 양은  $A_{r,c} - (A_{r,c}/5) \times (\text{확산된 방향의 개수})$  이다.

→ 미세먼지 확산을 구현할 때 유의할 점이 있을까?

## 미세먼지 안녕!

- 1초 동안 아래 적힌 일이 순서대로 일어난다.
- 미세먼지가 확산된다. 확산은 미세먼지가 있는 모든 칸에서 동시에 일어난다.
  - (r, c)에 있는 미세먼지는 **인접한 네 방향으로 확산된다.**
  - 인접한 방향에 **공기청정기**가 있거나, 칸이 없으면 그 방향으로 **확산이 일어나지 않는다.**
  - 확산되는 양은  $A_{r,c}/5$ 이고 **소수점은 버린다.**
  - (r, c)에 남은 미세먼지의 양은  $A_{r,c} - (A_{r,c}/5) \times (\text{확산된 방향의 개수})$  이다.

→ 미세먼지 확산을 구현할 때 유의할 점이 있을까?

6		
4	8	4



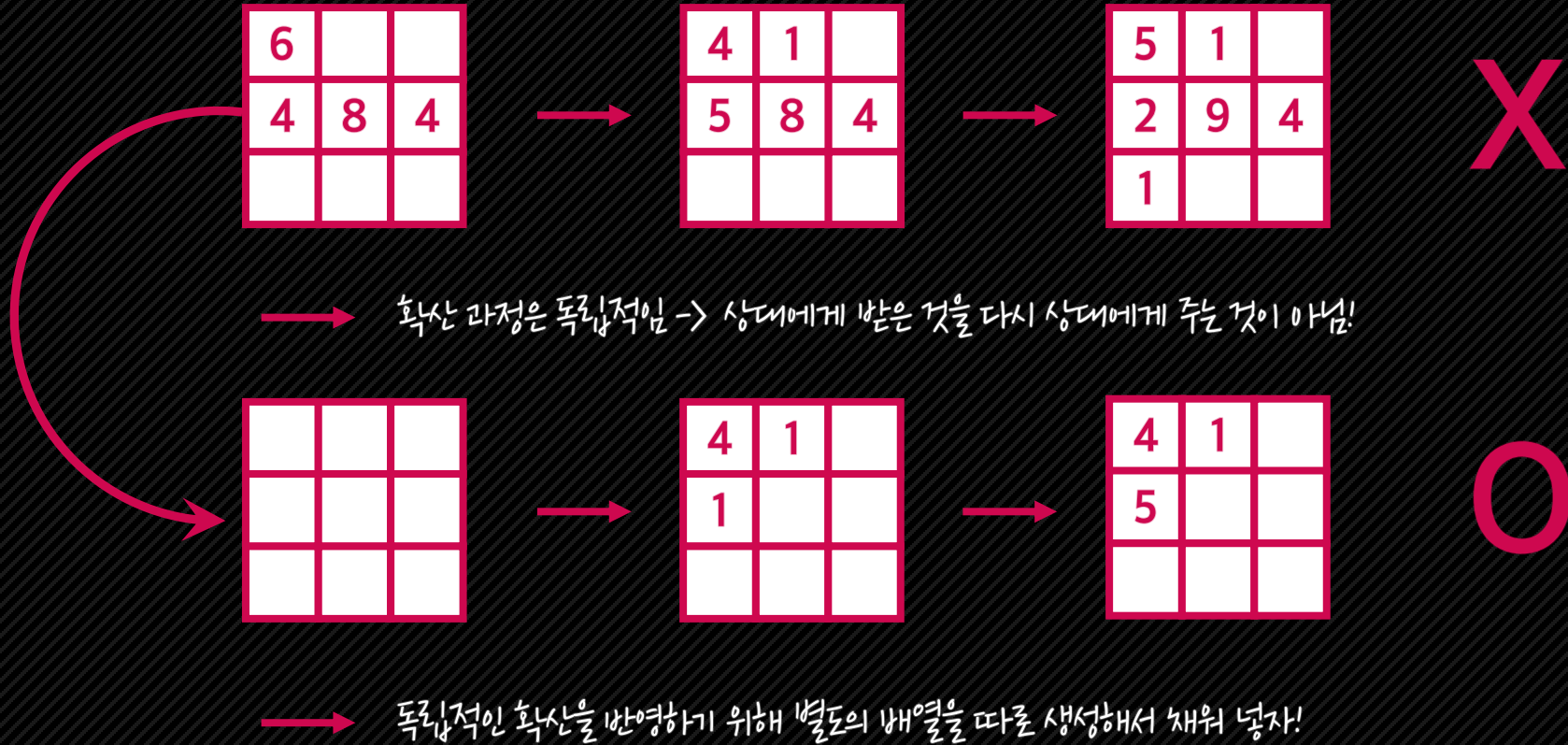
4	1	
5	8	4



5	1	
2	9	4
1		

?

## 미세먼지 안녕!





## 미세먼지 안녕!

- 1초 동안 아래 적힌 일이 순서대로 일어난다.
- 공기청정기가 작동한다. 공기청정기에서는 바람이 나온다.  
위쪽 공기청정기의 바람은 반시계방향으로 순환하고, 아래쪽 공기청정기의 바람은 시계방향으로 순환한다.  
바람이 불면 미세먼지가 바람의 방향으로 모두 한 칸씩 이동한다.  
공기청정기에서 부는 바람은 미세먼지가 없는 바람이고, 공기청정기로 들어간 미세먼지는 모두 정화된다.

## 미세먼지 안녕!

- 1초 동안 아래 적힌 일이 순서대로 일어난다.
  - 공기청정기가 작동한다. 공기청정기에서는 바람이 나온다.  
위쪽 공기청정기의 바람은 반시계방향으로 순환하고, 아래쪽 공기청정기의 바람은 시계방향으로 순환한다.  
바람이 불면 미세먼지가 바람의 방향으로 모두 한 칸씩 이동한다.  
공기청정기에서 부는 바람은 미세먼지가 없는 바람이고, 공기청정기로 들어간 미세먼지는 모두 정화된다.
- 이동 방향을 고려하면... 윗부분의 이동과 아랫부분의 이동을 따로 구현하는게 좋을 것 같다.

미세먼지 안녕!

## 문제 분석

- 데이터 입력 받기
- 공기청정기의 위치 확인
- 미세먼지 확산
- 공기청정기 작동
- 최종 결과 출력

# 미세먼지 안녕!

## 문제 분석

- 데이터 입력 받기
- 공기청정기의 위치 확인
- 미세먼지 확산
  - 독립적인 확산을 반영하기 위해 별도의 배열을 따로 생성해서 채워 넣자!
- 공기청정기 작동
  - 윗부분과 아랫부분의 이동을 따로 구현하자.
- 최종 결과 출력

# 미세먼지 안녕!

## 코드 구조 및 설계

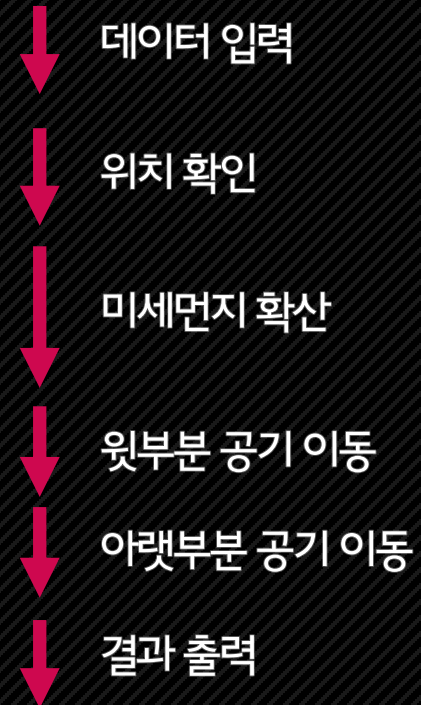
- 데이터 입력 받기
- 공기청정기의 위치 확인
- 미세먼지 확산

→ 독립적인 확산을 반영하기 위해 별도의 배열을 따로 생성해서 채워 넣자!

- 공기청정기 작동

→ 윗부분과 아랫부분의 이동을 따로 구현하자.

- 최종 결과 출력



# 미세먼지 안녕!

## 코드 구조 및 설계

```
def check(x, y, tmp):  
    # 상하좌우 체크해서 데이터 추가 후, 몇 번 이동했는지 출력
```

```
def spread():  
    # 임시 배열 생성  
    # 먼지가 있으면 check() 실행  
    # 값 실제 배열로 복사
```

```
def cleanUp():
```

```
def cleanDown():
```

```
def solve():  
    spread()  
    cleanUp()  
    cleanDown()
```

```
def find_air():
```

```
if __name__ == "__main__":  
    # 데이터 입력  
    find_air()  
    for i in range(T):  
        solve()
```



**잠깐!**

뒤로 넘어가기 전,  
딱 **5분만** 직접 문제를 읽고 분석해봅시다.



- 한 야구팀의 감독 아인타는 타순을 정하려고 한다.
- 아인타 팀의 선수는 총 9명이 있고, 1번부터 9번까지 번호가 매겨져 있다.
- 아인타는 자신이 가장 좋아하는 선수인 1번 선수를 4번 타자로 미리 결정했다.
- 이제 다른 선수의 타순을 모두 결정해야 한다.
- 각 선수가 각 이닝에서 어떤 결과를 얻는지 미리 알고 있을 때, 가장 점수를 많이 얻는 타순의 득점을 구해보자.



다만...



아인타는 자신이 가장 좋아하는 선수인 1번 선수를 4번 타자로 미리 결정했다.

→ 일단 1번 선수는 빼고 섞은 다음에, 1번을 4번 위치에 가도록 수정만 하자.



## 문제 분석

- 데이터 입력 받기
- 모든 순서 탐색
- 각 순서에 대하여 점수 계산
- 최종 결과 출력



## 문제 분석

- 데이터 입력 받기

- 모든 순서 탐색

→ 일단 1번 선수는 빼고 섞은 다음에, 1번을 4번 위치에 가도록 수정만 하자.

- 각 순서에 대하여 점수 계산

→ 점수는 어떻게 계산할까? (hint: 1~3루 배열을 만들어보자!)

→ 아웃 3개면 이닝 종료, 이닝의 횟수가 N에 도달하면 게임이 종료되는 것 체크!

- 최종 결과 출력



## 코드 구조 및 설계

- 데이터 입력 받기

- 모든 순서 탐색

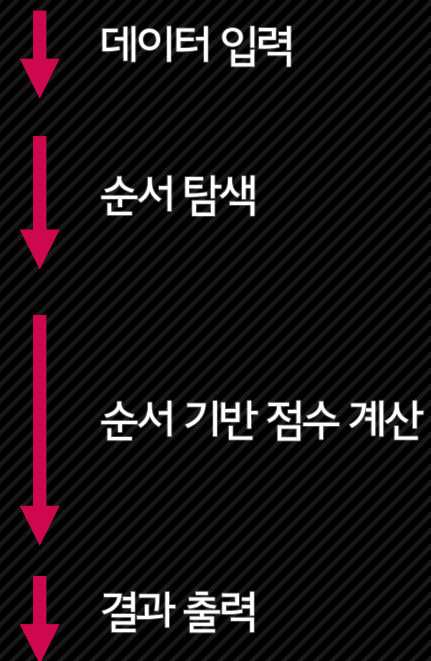
→ 일단 1번 선수는 빼고 섞은 다음에, 1번을 4번 위치에 가도록 수정만 하자.

- 각 순서에 대하여 점수 계산

→ 점수는 어떻게 계산할까? (hint: 1~3루 배열을 만들어보자!)

→ 아웃 3개면 이닝 종료, 이닝의 횟수가 N에 도달하면 게임이 종료되는 것 체크!

- 최종 결과 출력









## 독특한 계산기



### 도전!

지금까지 활용한 문제 분석 방법을 토대로  
직접 문제를 풀어봅시다!

끝난게 아니다!

-  Gold 3 - 청소년 상어 (#19236, 삼성 SW 역량테스트 A형 유사문항)
-  Gold 2 - 2048 (Easy) (#12100, 삼성 SW 역량테스트 A형 유사문항)
-  Gold 2 - 구슬탈출 2 (#13460, 삼성 SW 역량테스트 A형 유사문항)
-  Gold 2 - 모노미노도미노 2 (#20061, 삼성 SW 역량테스트 A형 유사문항)
-  Level 3 - 자물쇠와 열쇠 (kakao 2020 Blind Recruitment 1차 코딩 테스트)
-  Level 3 - 기둥과 보 설치 (kakao 2020 Blind Recruitment 1차 코딩 테스트)

- 구현은 정말 많이 나오는 유형인 만큼, 꾸준히 연습해야 합니다...
- 해당 문제에 대한 풀이는 매 수업시간 마다 1문제 씩 진행할 예정입니다.

“Any question?”