


# 구현 - 12장

카카오 해설 및 출제의도

## 2020 신입 개발자 블라인드 채용 1차 코딩 테스트 문제 해설

올해에도 2020이라는 멋진 숫자와 함께, 카카오의 신입 개발자 채용을 시작했습니다! 그 여정 중 첫 단계로 1차 코딩 테스트가 지난 9월 7일 토요일 오후 2시부터 오후 7시까지 5시간 동안 진행됐는데요. 저희 준비위원들도 설렘과 긴장 속에 원활한 진행을 위해 노력했고, 무사히 1차 테스트를 마칠 수 있었습니다. 테스트에는 총 7문

 <https://tech.kakao.com/2019/10/02/kakao-blind-recruitment-2020-round1/>



## 1. 럭키 스트레이트(해결)

게임의 아웃복서 캐릭터는 필살기인 '럭키 스트레이트' 기술이 있다. 이 기술은 매우 강력하기 때문에 게임 내 점수가 특정 조건을 만족할 때만 사용이 가능하다.

특정 조건이란 현재 캐릭터의 점수가 N이라고 할 때 자릿수를 기준으로 점수 N을 반으로 나눠 왼쪽 부분의 합과 오른쪽 부분의 합이 같은 상황을 의미한다. 예를 들어 현재 점수 N이 123,402 이라면 왼쪽, 오른쪽의 합이 모두 6이기 때문에 스킬 사용이 가능하다.

현재 점수 N이 주어질 때, 스킬을 사용할 수 있는 상태인지 아닌지 구하여라. 단, N의 자릿수는 무조건 짝수이다.



해결

N을 배열로 만든 후, 중간 인덱스를 잡아 왼쪽의 합, 오른쪽의 합을 구해 비교하고 결과를 출력한다.



코드

```
import sys

N = list(str(sys.stdin.readline().strip()))

midIdx = len(N) // 2

leftSum = 0
rightSum = 0

for i in range(midIdx):
    leftSum += int(N[i])
for i in range(midIdx, len(N)):
    rightSum += int(N[i])

if leftSum == rightSum:
    print("LUCKY")
else:
    print("READY")
```

## 2. 문자열 재정렬(해결)

알파벳 대문자와 숫자로만 구성된 문자열이 입력으로 주어진다. 이때 모든 알파벳을 오름차순으로 정렬하여 이어 출력한 후, 그 뒤에 모든 숫자를 더한 값을 이어서 출력한다.



해결

알파벳과 숫자를 각각 다른 배열에 저장한 후, 알파벳 정렬, 숫자 합을 구해 그대로 출력한다.



코드

```
# 내 정답
data = list(input())

strs = [x for x in data if x >= 'A' and x <= 'Z']
nums = [int(x) for x in data if x >= '0' and x <= '9']

strs.sort()

print(''.join(strs) + str(sum(nums)))

# 정답 코드
data = list(input())

strs = [x for x in data if x.isalpha()]
nums = [int(x) for x in data if not x.isalpha()]

strs.sort()

print(''.join(strs) + str(sum(nums)))
```

### 3. 문자열 압축 (정답률 25.9%) (미해결)

데이터 처리 전문가가 되고 싶은 “어피치”는 문자열을 압축하는 방법에 대해 공부하고 있습니다. 최근에 대량의 데이터 처리를 위한 간단한 비손실 압축 방법에 대해 공부를 하고 있는데, 문자열에서 같은 값이 연속해서 나타나는 것을 그 문자의 개수와 반복되는 값으로 표현하여 더 짧은 문자열로 줄여서 표현하는 알고리즘을 공부하고 있습니다.

간단한 예로 ‘aabbaccc’의 경우 ‘2a2ba3c’(문자가 반복되지 않아 한 번만 나타난 경우 1은 생략)와 같이 표현할 수 있는데, 이러한 방식은 반복되는 문자가 적은 경우 압축률이 낮다는 단점이 있습니다. 예를 들어 ‘abcabcdede’와 같은 문자열은 전혀 압축되지 않겠조. 어피치는 이러한 단점을 해결하기 위해 문자열을 1개 이상의 단위로 잘라서 압축하여 더 짧은 문자열로 표현할 수 있는지 방법을 찾아보려고 합니다.

예를 들어, ‘ababacdcdababacdcd’의 경우 문자를 1개 단위로 자르면 전혀 압축되지 않지만, 2개 단위로 자르면 ‘2ab2cd2ab2cd’가 되고, 8개 단위로 자르면 ‘2ababacdcd’가 되어 가장 짧게 압축할 수 있습니다.

압축할 문자열 s가 매개변수로 주어질 때, 위에 설명한 방법으로 1개 이상 단위로 문자열을 잘라 압축하여 표현한 문자열 중 가장 짧은 것의 길이를 리턴하도록 solution 함수를 완성해주세요.



해결

#### 1. 내 해결

- ... 강 모르겠는데... 문자열 길이가 1000 이하이고 시간 제한이 1초라  $O(N^2)$ 까지는 써도 될 것 같다 정도 알겠네.. 1~문자열 길이 까지의 단위로 잘랐을 때 문자열의 길이를 배열에 저장하고 그 배열 중 최소값을 구하면 되지 않을까 싶기는 하다. 근데 이걸 어떻게 구현하냐고 ㅎㅎ

## 4. 자물쇠와 열쇠 (정답률 7.4%) (미해결)

고고학자인 튜브는 고대 유적지에서 보물과 유적이 가득할 것으로 추정되는 비밀의 문을 발견하였습니다. 그런데 문을 열려고 살펴보니 특이한 형태의 자물쇠로 잠겨 있었고 문 앞에는 특이한 형태의 열쇠와 함께 자물쇠를 푸는 방법에 대해 다음과 같이 설명해주는 종이가 발견되었습니다.

잠겨있는 자물쇠는 격자 한 칸의 크기가  $1 \times 1$ 인  $N \times N$  크기의 정사각 격자 형태이고 특이한 모양의 열쇠는  $M \times M$  크기인 정사각 격자 형태로 되어 있습니다.

자물쇠에는 홈이 파여 있고 열쇠 또한 홈과 돌기 부분이 있습니다. 열쇠는 회전과 이동이 가능하며 열쇠의 돌기 부분을 자물쇠의 홈 부분에 딱 맞게 채우면 자물쇠가 열리게 되는 구조입니다. 자물쇠 영역을 벗어난 부분에 있는 열쇠의 홈과 돌기는 자물쇠를 여는 데 영향을 주지 않지만, 자물쇠 영역 내에서는 열쇠의 돌기 부분과 자물쇠의 홈 부분이 정확히 일치해야 하며 열쇠의 돌기와 자물쇠의 돌기가 만나서는 안됩니다. 또한 자물쇠의 모든 홈을 채워 비어있는 곳이 없어야 자물쇠를 열 수 있습니다.

열쇠를 나타내는 2차원 배열 `key`와 자물쇠를 나타내는 2차원 `lock`이 매개변수로 주어질 때, 열쇠로 자물쇠를 열 수 있으면 `true`를, 열 수 없으면 `false`를 return 하도록 `solution` 함수를 완성하세요.



#### 해결

#### 1. 내 해결

주어진 열쇠에 대해 상하좌우로 0~M만큼 움직인 값들을 저장하고 해당 값들을 회전시켜 가며 자물쇠와 일치하는지 확인

#### 2. 정답

대체 3배로 늘릴 생각을 어떻게 하냐고... 상하좌우 다 맞추려면 3배로 해야되는건 해설 보니까 알겠는데 참... 이걸 어떻게 아는지



#### 코드

#### 1. 내 코드 - 39점 - 그냥 True, False라서 39점 같기도...

```
import copy

def solution(key, lock):
    M, N = len(key), len(lock)

    answer = False

    # 자물쇠 빈 칸의 위치 좌표
    lock_locs = []
```

```

for i in range(N):
    for j in range(N):
        if lock[i][j] == 0:
            lock_locs.append([i, j])

# 상하좌우로 0~M만큼 움직여서 모든 데이터를 move_keys에 저장
# move_keys는 3차원 배열이 됨
move_keys = []

move_key = []
for i in range(M):
    for j in range(M):
        if key[i][j] == 1:
            move_key.append([i, j])

# 처음 움직이지 않았을 때의 좌표
move_keys.append(move_key)

# 네 방향으로 이동하면서 모든 데이터 저장
for i in range(4):
    # 첫 위치좌표 복사
    tmp_keys = copy.deepcopy(move_key)

    # M번만큼 이동
    for _ in range(M):
        # 첫 위치 좌표의 모든 키들에 대해서
        for tmp_key in tmp_keys:
            if i == 0: # 상
                tmp_key[0] -= 1
            elif i == 1: # 하
                tmp_key[0] += 1
            elif i == 2: # 좌
                tmp_key[1] -= 1
            else: # 우
                tmp_key[1] += 1

        move_keys.append(tmp_keys)

# 회전 없이 이동만 했던 모든 키들에 대해서
for move_key in move_keys:
    tmp_keys = copy.deepcopy(move_key) # 복사 후
    rotate_keys = [] # 4번 회전한 데이터들 # 3차원 배열
    rotate_keys.append(tmp_keys) # 처음 회전 안 했을 경우 추가

    # 3번 회전
    for i in range(3):
        rotate_key = []

        for tmp_key in tmp_keys:
            # 회전 후의 행 = 회전 전의 열, 회전 후의 열 = key 전체 배열의 마지막 인덱스 - 회전 전의 행
            x, y = tmp_key[1], M - 1 - tmp_key[0]
            rotate_key.append([x, y])

        rotate_keys.append(rotate_key)
        tmp_keys = copy.deepcopy(rotate_key)

# 모든 회전한 데이터들에 대해
for rotate_key in rotate_keys:
    flag = True # 자물쇠와 열쇠가 맞는지에 대한 플래그

    # 자물쇠의 모든 좌표에 대해
    for lock_loc in lock_locs:
        # 자물쇠의 각 좌표가 회전한 데이터에 없으면 -> 맞지 않는 열쇠인 경우
        if lock_loc not in rotate_key:
            flag = False
            break # 종료

    # 자물쇠와 열쇠가 맞을 경우
    if flag:
        more_flag = True

        # 자물쇠랑 돌기랑 겹치는 것 확인
        for key in rotate_key:
            if key[0] >= 0 and key[1] >= 0 and key[0] < M and key[1] < M and key not in lock_locs:
                more_flag = False

        # 자물쇠 돌기랑 열쇠가 겹치는 경우가 없을 경우
        if more_flag:
            answer = True
            break

# answer가 True면 종료

```

```

# 종료 안하면 다음 데이터에 대한 계산을 다시 하기 때문에 False로 바뀜
if answer:
    break

return answer

```

## 2. 정답

```

# 2차원 리스트 90도 회전
def rotate_a_matrix_by_90_degree(a):
    n = len(a) # 행 길이
    m = len(a[0]) # 열 길이

    result = [[0] * n for _ in range(m)]

    for i in range(n):
        for j in range(m):
            result[j][n - i - 1] = a[i][j]

# 자물쇠의 중간 부분이 모두 1인지 확인
def check(new_lock):
    lock_length = len(new_lock) // 3

    for i in range(lock_length, lock_length * 2):
        for j in range(lock_length, lock_length * 2):
            if new_lock[i][j] != 1:
                return False

    return True

def solution(key, lock):
    n = len(lock)
    m = len(key)

    # 자물쇠 크기를 기존의 3배로
    new_lock = [[0] * (n * 3) for _ in range(n * 3)]

    # 새로운 자물쇠의 중앙 부분에 기존의 자물쇠 값 넣기
    for i in range(n):
        for j in range(n):
            new_lock[i + n][j + n] = lock[i][j]

    # 4가지 방향에 대해 확인
    for rotation in range(4):
        key = rotate_a_matrix_by_90_degree(key) # 열쇠 회전

        for x in range(n * 2):
            for y in range(n * 2):
                for i in range(m):
                    for j in range(m):
                        new_lock[x + i][y + j] += key[i][j]

                if check(new_lock):
                    return True

                for i in range(m):
                    for j in range(m):
                        new_lock[x + i][y + j] -= key[i][j]

    return False

```

## 5. 뱀 (해결)

'Dummy' 라는 도스게임이 있다. 이 게임에는 뱀이 나와서 기어다니는데, 사과를 먹으면 뱀 길이가 늘어난다. 뱀이 이리저리 기어다니다가 벽 또는 자기자신의 몸과 부딪히면 게임이 끝난다.

게임은  $N \times N$  정사각 보드위에서 진행되고, 몇몇 칸에는 사과가 놓여져 있다. 보드의 상하좌우 끝에 벽이 있다. 게임이 시작할때 뱀은 맨위 맨좌측에 위치하고 뱀의 길이는 1 이다. 뱀은 처음에 오른쪽을 향한다.

뱀은 매 초마다 이동을 하는데 다음과 같은 규칙을 따른다.

- 먼저 뱀은 몸길이를 늘려 머리를 다음칸에 위치시킨다.
- 만약 이동한 칸에 사과가 있다면, 그 칸에 있던 사과가 없어지고 꼬리는 움직이지 않는다.
- 만약 이동한 칸에 사과가 없다면, 몸길이를 줄여서 꼬리가 위치한 칸을 비워준다. 즉, 몸길이는 변하지 않는다.

사과의 위치와 뱀의 이동경로가 주어질 때 이 게임이 몇 초에 끝나는지 계산하라.



#### 해결

시뮬레이션 문제이다. 뱀의 꼬리를 큐에 넣어 푸는 방법이 맞아 보인다.



#### 코드

##### 1. 내가 생각한 코드

```
import sys
from collections import deque

def changeDir(cur, dir): # 방향 전환
    if dir == 'r':
        if cur == 3:
            return 1
        elif cur == 2:
            return 0
        elif cur == 1:
            return 2
        else:
            return 3
    else:
        if cur == 3:
            return 0
        elif cur == 2:
            return 1
        elif cur == 1:
            return 3
        else:
            return 2

N = int(sys.stdin.readline()) # 보드 크기
K = int(sys.stdin.readline()) # 사과 개수

map = [[0] * (N + 1) for _ in range(N + 1)] # 맵

for _ in range(K):
    x, y = sys.stdin.readline().strip().split()
    map[int(x)][int(y)] = 2 # 사과 위치 = 2

L = int(sys.stdin.readline()) # 뱀의 방향 변환 횟수

change_dir = [] # 뱀의 방향 변환 저장할 배열

for _ in range(L):
    x, c = sys.stdin.readline().strip().split()
    change_dir.append((x, c))
```

```

sec = 0 # 시간

snake_info = deque([])
snake_info.append((1, 1))

x, y = 1, 1 # 머리 위치
map[1][1] = 1 # 뱀 위치 = 1

snake_dir = 3 # 뱀의 머리가 현재 바라보는 방향 # default 오른쪽

dir = [(-1, 0), (1, 0), (0, -1), (0, 1)] # 상하좌우

while True:
    sec += 1 # 1초 증가

    # 머리 이동 전 체크
    dx = x + dir[snake_dir][0]
    dy = y + dir[snake_dir][1]

    # 벽에 닿았거나 본인의 몸과 부딪히면 종료
    if dx > N or dy > N or dx < 1 or dy < 1 or map[dx][dy] == 1:
        break

    # 꼬리 이동
    # 머리가 사과를 안먹었을 경우
    if map[dx][dy] != 2:
        tail = snake_info.popleft()
        t_x = tail[0]
        t_y = tail[1]

        map[t_x][t_y] = 0

    # 머리 이동
    map[dx][dy] = 1
    x, y = dx, dy
    # 뱀 위치 정보 갱신
    snake_info.append((dx, dy))

    # 뱀의 이동 경로가 바뀌었을 경우
    if len(change_dir) > 0 and sec == int(change_dir[0][0]):
        # 오른쪽으로 90도
        if change_dir[0][1] == 'D':
            snake_dir = changeDir(snake_dir, 'r')
        else: # 왼쪽으로 90도
            snake_dir = changeDir(snake_dir, 'l')

        change_dir.pop(0)

print(sec)

```

## 2. 정답

```

n = int(input())
k = int(input())
data = [[0] * (n + 1) for _ in range(n + 1)] # 맵 정보
info = [] # 방향 회전 정보

# 맵 정보(사과 있는 곳은 1)
for _ in range(k):
    a, b = map(int, input().split())
    data[a][b] = 1

# 방향 회전 정보 입력
l = int(input())
for _ in range(l):
    x, c = input().split()
    info.append((int(x), c))

dx = [0, 1, 0, -1]
dy = [1, 0, -1, 0]

def turn(direction, c):
    if c == "L":
        direction = (direction - 1) % 4
    else:
        direction = (direction + 1) % 4

```

```

return direction

def simulate():
    x, y = 1, 1 # 뱀의 머리 위치
    data[x][y] = 2 # 뱀이 존재하는 위치는 2로 표시
    direction = 0 # 처음은 동쪽 보는 중
    time = 0
    index = 0 # 다음에 회전할 정보
    q = [(x, y)] # 뱀이 차지하고 있는 위치 정보(꼬리가 앞쪽)

    while True:
        nx = x + dx[direction]
        ny = y + dy[direction]

        # 죽지 않는 경우라면
        if 1 <= nx and nx <= n and 1 <= ny and ny <= n and data[nx][ny] != 2:
            # 이동한 곳에 사과가 없다면
            if data[nx][ny] == 0:
                # 이동
                data[nx][ny] = 2
                q.append((nx, ny))
                # 꼬리 제거
                px, py = q.pop()
                data[px][py] = 0

            # 사과가 있다면
            if data[nx][ny] == 1:
                # 이동만
                data[nx][ny] = 2
                q.append((nx, ny))

        # 벽이나 자신에게 부딪히면
        else:
            time += 1
            break

        x, y = nx, ny # 머리를 실제로 이동

        time += 1
        if index < l and time == info[index][0]: # 회전할 시간인 경우 회전
            direction = turn(direction, info[index][1])
            index += 1

    return time

print(simulate())

```

## 6. 기둥과 보 설치 (정답률 1.9%) (미해결)

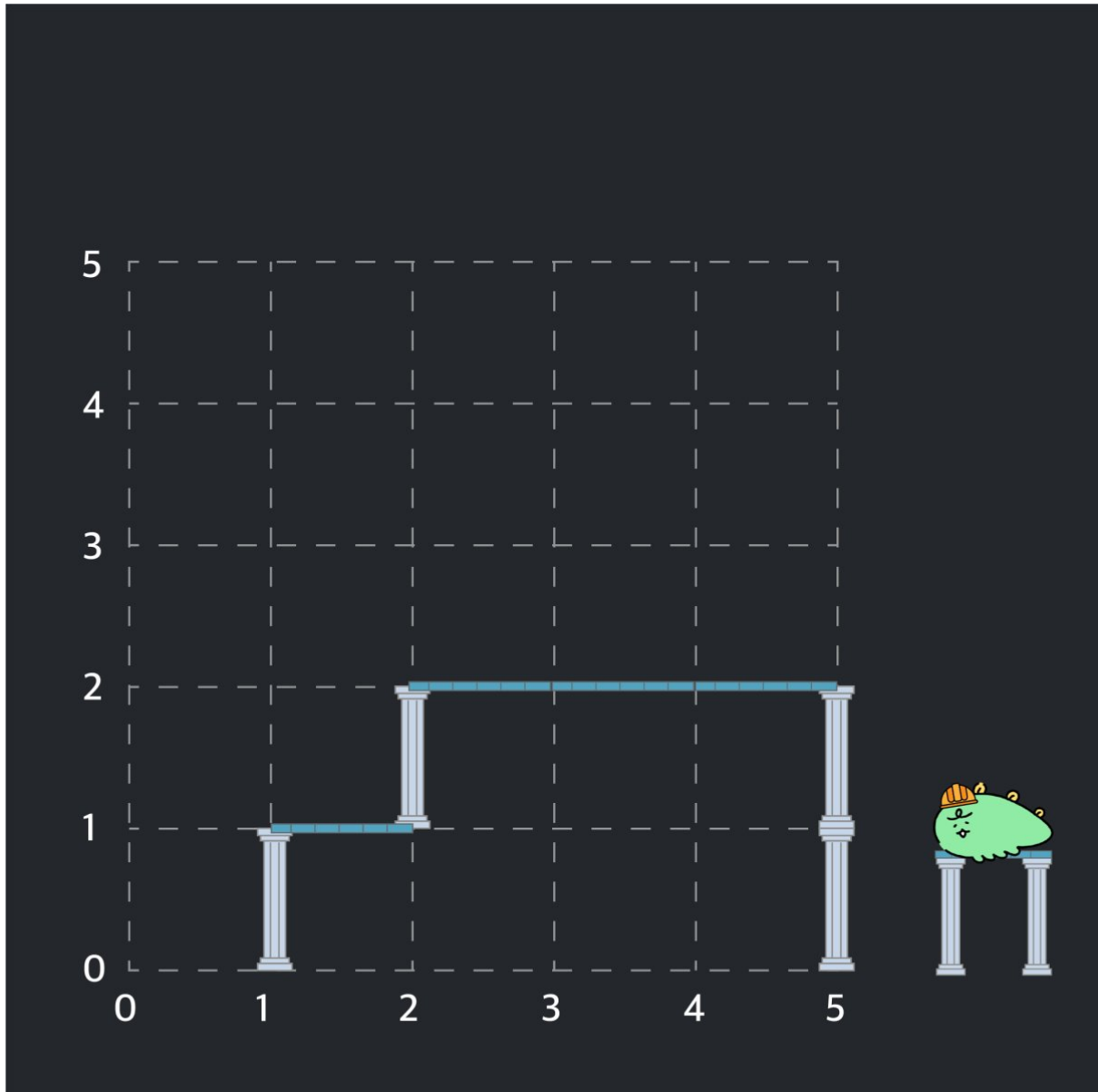
빙하가 깨지면서 스노우타운에 떠나려 온 "쵸르디"는 인생 2막을 위해 주택 건축사업에 뛰어들기로 결심하였습니다. "쵸르디"는 **기둥과 보**를 이용하여 벽면 구조물을 자동으로 세우는 로봇을 개발할 계획인데, 그에 앞서 로봇의 동작을 시뮬레이션 할 수 있는 프로그램을 만들고 있습니다. 프로그램은 **2차원 가상 벽면**에 기둥과 보를 이용한 구조물을 설치할 수 있는데, 기둥과 보의 **길이가 1인 선분**으로 표현되며 다음과 같은 규칙을 가지고 있습니다.

- 기둥은 바닥 위에 있거나 보의 한쪽 끝 부분 위에 있거나, 또는 다른 기둥 위에 있어야 합니다.
- 보의 한쪽 끝 부분이 기둥 위에 있거나, 또는 양쪽 끝 부분이 다른 보와 동시에 연결되어 있어야 합니다.

단, 바닥은 벽면의 맨 아래 지면을 말합니다.

2차원 벽면은 **n x n** 크기 정사각 격자 형태이며, 각 격자는 **1 x 1** 크기입니다. 맨 처음 벽면은 비어있는 상태입니다. 기둥과 보의 격자선의 교차점에 걸치지 않고, 격자 칸의 각 변에 정확히 일치하도록 설치할 수 있습니다. 다음은 기둥과 보를 설치해 구조물을 만든 예시입니다.





예를 들어, 위 그림은 다음 순서에 따라 구조물을 만들었습니다.

1. (1, 0)에서 위쪽으로 기둥을 하나 설치 후, (1, 1)에서 오른쪽으로 보를 하나 만듭니다.
2. (2, 1)에서 위쪽으로 기둥을 하나 설치 후, (2, 2)에서 오른쪽으로 보를 하나 만듭니다.
3. (5, 0)에서 위쪽으로 기둥을 하나 설치 후, (5, 1)에서 위쪽으로 기둥을 하나 더 설치합니다.
4. (4, 2)에서 오른쪽으로 보를 설치 후, (3, 2)에서 오른쪽으로 보를 설치합니다.

만약 (4, 2)에서 오른쪽으로 보를 먼저 설치하지 않고, (3, 2)에서 오른쪽으로 보를 설치하려 한다면 2번 규칙에 맞지 않으므로 설치가 되지 않습니다. 기둥과 보를 삭제하는 기능도 있는데 기둥과 보를 삭제한 후에 남은 기둥과 보들 또한 위 규칙을 만족해야 합니다. 만약, 작업을 수행한 결과가 조건을 만족하지 않는다면 해당 작업은 무시됩니다.

벽면의 크기  $n$ , 기둥과 보를 설치하거나 삭제하는 작업이 순서대로 담긴 2차원 배열 `build_frame`이 매개변수로 주어질 때, 모든 명령어를 수행한 후 구조물의 상태를 `return` 하도록 `solution` 함수를 완성해주세요.



## 해결

시뮬레이션 문제이기에 해결은 없다.



## 코드

### 1. 내 코드

```

# 기둥 설치
def set_column(n, arr, x, y):
    if y + 1 <= n: # 기둥 설치 좌표 평면 안에서 실행될 경우
        if y == 0 or arr[x][y] == 2 or arr[x][y] == 1: # 바닥이거나 해당 좌표에 보가 있거나 기둥 위거나
            arr[x][y] = 1
            arr[x][y + 1] = 1

    return arr

# 기둥 삭제
def del_column(arr, x, y):
    return arr

# 보 설치
def set_girder(n, arr, x, y):
    if x + 1 <= n:
        if arr[x][y] == 1 or (arr[x][y] == 2 and arr[x + 1][y] == 2):
            arr[x][y] = 2
            arr[x + 1][y] = 2

    return arr

# 보 삭제
def del_girder(arr, x, y):
    return arr

def solution(n, build_frame):
    answer = []
    # 맵 만들기
    # 교차점에 대한 정보를 가져야 되기 때문에 크기가 1만큼 더 커야됨
    map = [[0] * (n + 1) for _ in range(n + 1)]

    # 각각의 build_frame에 대해
    for step in build_frame:
        x, y, a, b = step[0], step[1], step[2], step[3]

        if a == 0: # 기둥일 경우
            if b == 0: # 삭제의 경우
                del_column(map, x, y)
            else: # 추가의 경우
                set_column(n + 1, map, x, y)
        else: # 보의 경우
            if b == 0: # 삭제의 경우
                del_girder(map, x, y)
            else: # 추가의 경우
                set_girder(n + 1, map, x, y)

    # 맵의 모든 정보에 대해
    for i in range(n + 1):
        for j in range(n + 1):
            if map[i][j] != 0: # 값이 0이 아니면(기둥 또는 보가 설치되어 있으면)
                result = [i, j, map[i][j] - 1] # [x, y, 건축물]
                answer.append(result) # 정답에 추가

    # x, y 순서로 오름차순 정렬
    answer = sorted(answer, key = lambda x: (x[0], x[1]))

    return answer

```

작성하다가 기둥과 보의 삭제에서 막혔다. 내 코드대로 하면 기둥과 보의 설치 순서 또한 중요하게 된다. 이 설치 순서를 삭제에서 전부 부담해서 체크해야 되는데... 못하겠다. 이 방법이 아닌가보다

## 2. 정답 코드

```
# 현재 설치된 구조물이 가능한 구조물인지 확인하는 함수
def possible(answer):
    for x, y, stuff in answer:
        if stuff == 0: # 기둥일 경우
            # 바닥 위 혹은 보의 한쪽 끝부분 위 혹은 다른 기둥 위라면 정상
            if y == 0 or [x - 1, y, 1] in answer or [x, y, 1] in answer or [x, y - 1, 0] in answer:
                continue
            # 아니라면
            return False
        elif stuff == 1: # 보인 경우
            # 한쪽 끝부분이 기둥 위 혹은 양쪽 끝부분이 다른 보와 동시에 연결이라면 정상
            if [x, y - 1, 0] in answer or [x + 1, y - 1, 0] in answer or ([x - 1, y, 1] in answer and [x + 1, y, 1] in answer):
                continue
            # 아니라면
            return False
    return True

def solution(n, build_frame):
    answer = []

    for frame in build_frame:
        x, y, stuff, operate = frame

        if operate == 0: # 삭제의 경우
            answer.remove([x, y, stuff])

            if not possible(answer):
                answer.append([x, y, stuff])

        if operate == 1: # 설치의 경우
            answer.append([x, y, stuff])

            if not possible(answer):
                answer.remove([x, y, stuff])

    return sorted(answer)
```

## 7. 치킨 배달 (해결)

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 “**치킨거리**”라는 말을 주로 사용한다. **치킨 거리**는 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.

임의의 두 칸  $(r1, c1)$ 과  $(r2, c2)$  사이의 거리는  $|r1-r2| + |c1-c2|$ 로 구한다. 즉, 행간 거리 + 열간 거리이다.

이 도시에 있는 치킨집은 모두 같은 프랜차이즈이다. 프랜차이즈 본사에는 수익을 증가시키기 위해 일부 치킨집을 폐업시키려고 한다. 오랜 연구 끝에 이 도시에서 가장 수익을 많이 낼 수 있는 치킨집의 개수는 최대  $M$ 개라는 사실을 알아냈다.

도시에 있는 치킨집 중에서 최대  $M$ 개를 고르고, 나머지 치킨집은 모두 폐업시켜야 한다. 어떻게 고르면, 도시의 치킨 거리가 가장 작게 될지 구하는 프로그램을 작성하라.



## 해결

### 1. 내가 생각한 해결

- 치킨집을 1~M개 선택하는 모든 조합에 대해 각 집에서 가장 가까운 집까지의 거리의 합을 구해 최소값을 출력한다.

### 2. 정답

- 1~M개가 아니라 M개만 고려해도 된다.. 만약 최대 3개를 고르라는 경우에서 1개의 치킨집 주위에만 집이 다 있다고 해도 3개를 선택할 때 그 1개의 치킨집이 꺼 있으면 최소값은 똑같다.



## 코드

```
# 내 정답
# 정답 코드를 보니... 굳이 1~M개를 선택하지 말고 M개에 대한 조합만 생각해도 된다...
import itertools
import sys

N, M = map(int, sys.stdin.readline().strip().split())

maps = []

for _ in range(N):
    maps.append(list(map(int, sys.stdin.readline().strip().split())))

chickens = []

for i in range(N):
    for j in range(N):
        if maps[i][j] == 2:
            chickens.append((i, j))

min_distance = 1e9

for i in range(1, M + 1): # 치킨집 1~M개를 선택
    for chicken in list(itertools.combinations(chickens, i)): # 치킨집 i개를 선택하는 모든 조합에 대해
        distance = 0

        for j in range(N):
            for k in range(N):
                if maps[j][k] == 1: # 각각 집에서
                    min_dis = abs(j - chicken[0][0]) + abs(k - chicken[0][1]) # 치킨 조합의 첫 번째 치킨집까지의 거리가 최소라고 해놓고

                    for data in chicken: # 치킨 조합의 모든 치킨집에 대해 집에서 치킨집까지의 최소 거리를 구하고
                        tmp_dis = abs(j - data[0]) + abs(k - data[1])
                        if tmp_dis < min_dis:
                            min_dis = tmp_dis

                    distance += min_dis # 최소 거리를 구해 거리에 더해준다.

        if distance < min_distance:
            min_distance = distance

print(min_distance)
```

- 위와 같이 1~M개 선택이면 엄청나게 느린 속도로 통과하지만... M개에 대해서만 하면 훨씬 빨라진다. 평균이 25%가 됐다.

## 8. 외벽 점검 (정답률 0.6%) (미해결)

레스토랑을 운영하고 있는 "스카피"는 레스토랑 내부가 너무 낡아 친구들과 함께 직접 리모델링하기로 했습니다. 레스토랑이 있는 곳은 스노우타운으로 매우 추운 지역이어서 내부 공사를 하는 도중에 주기적으로 외벽의 상태를 점검해야 할 필요가 있습니다.

레스토랑의 구조는 **완전히 동그란 모양**이고 **외벽의 총 둘레는 n미터**이며, 외벽의 몇몇 지점은 추위가 심할 경우 손상될 수도 있는 **취약한 지점들**이 있습니다. 따라서 내부 공사 도중에도 외벽의 취약 지점들이 손상되지 않았는 지, 주기적으로 친구들을 보내서 점검을 하기로 했습니다. 다만, 빠른 공사 진행을 위해 점검 시간을 1시간으로 제한했습니다. 친구들이 1시간 동안 이동할 수 있는 거리는 제각각이기 때문에, 최소한의 친구들을 투입해 취약 지점을 점검하고 나머지 친구들은 내부 공사를 돕도록 하려고 합니다. 편의 상 레스토랑의 정북 방향 지점을 0으로 나타내며, 취약 지점의 위치는 정북 방향 지점으로부터 시계 방향으로 떨어진 거리로 나타냅니다. 또, 친구들은 출발 지점부터 시계, 혹은 반시계 방향으로 외벽을 따라서만 이동합니다.

외벽의 길이 n, 취약 지점의 위치가 담긴 배열 weak, 각 친구가 1시간 동안 이동할 수 있는 거리가 담긴 배열 dist가 매개변수로 주어질 때, 취약 지점을 점검하기 위해 보내야 하는 친구 수의 최소값을 return 하도록 solution 함수를 완성해주세요.



## 해결

### 1. 내가 생각한 해결

- 각 취약점 사이의 거리를 구해 큐에 저장하고 내림차순 정렬한다.
- 취약점 사이의 거리가 가장 긴 것을 하나씩 제거하면서 반복문을 진행한다. 이때, 취약점 사이의 거리가 가장 긴 것을 하나씩 제거할 때마다 사람 수는 하나씩 늘어난다.
- 진행하다가 취약점 사이의 거리를 담은 배열에 남은게 없어졌으면 그냥 안되는 것이므로 -1을 리턴한다.

### 2. 정답

- dist의 크기가 8밖에 되지 않으므로 모든 경우의 수에 대해서 탐색을 진행한다.



## 코드

### 1. 내 코드(52점 나옴)

```
from collections import deque

def solution(n, weak, dist):
    answer = 0
    weak_dist = [] # 각 취약점 사이 거리

    dist.sort(reverse=True)

    for i in range(len(weak) - 1):
        weak_dist.append(weak[i + 1] - weak[i])

    # 맨 마지막 취약점과 맨 처음 취약점 사이 거리
    weak_dist.append((n - weak[-1]) + weak[0])

    # 거리 내림차순
    weak_dist.sort(reverse=True)
    weak_dist = deque(weak_dist)

    i = 1

    while True:
        wd_len = len(weak_dist)

        # 거리가 하나도 남지 않았으면
        if wd_len <= 0:
            answer = -1
            break

        # 최대값은 빼주기
        weak_dist.popleft()

        # 최대 친구들 i번째까지 가져오기
```

```

tmp_dist = [dist[i] for i in range(0, i)]

# 남은 거리 <= 친구들이 할 수 있는 값의 합
if sum(weak_dist) <= sum(tmp_dist):
    answer = i
    break
else:
    i += 1

return answer

```

## 2. 답

```

from itertools import permutations # 순열

def solution(n, weak, dist):
    # 길이를 2배로 늘려 원형을 일자 형태로 변경
    length = len(weak)
    for i in range(length):
        weak.append(weak[i] + n)
    answer = len(dist) + 1 # 투입할 친구 수의 최소값을 찾아야 하므로 나올 수 없는 값으로 초기화

    # 0부터 length - 1까지의 위치를 각각 시작점으로 설정
    for start in range(length):
        # 친구를 나열하는 모든 경우의 수 각각에 대해 확인
        for friends in list(permutations(dist, len(dist))):
            count = 1 # 투입할 친구 수
            position = weak[start] + friends[count - 1] # 해당 친구가 점검할 수 있는 마지막 위치

            for index in range(start, start + length): # 시작점부터 모든 취약 지점 확인
                if position < weak[index]: # 점검할 수 있는 위치를 벗어나는 경우
                    count += 1 # 새 친구 투입
                    if count > len(dist): # 더 투입 불가능하면 종료
                        break

            position = weak[index] + friends[count - 1]

            answer = min(answer, count)

    if answer > len(dist):
        return -1

    return answer

```