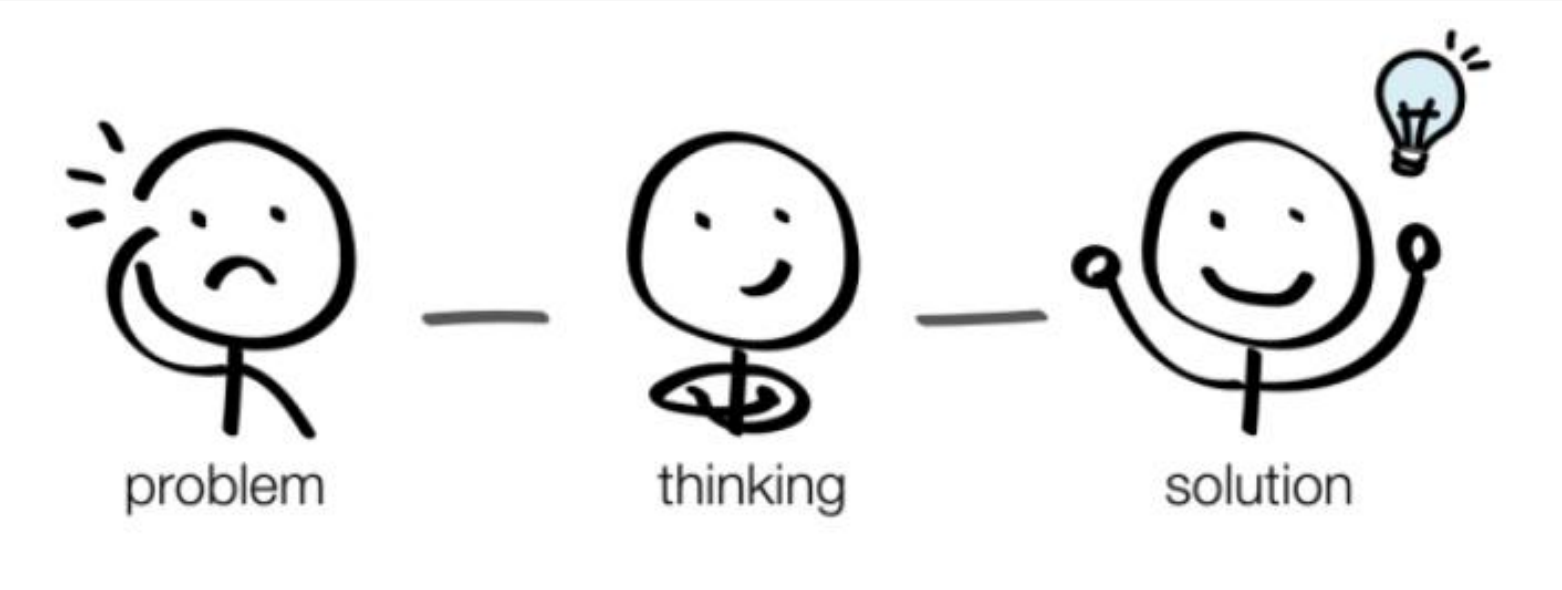
구현 알고리즘

01. 구현 알고리즘

구현 알고리즘 (Implementation Algorithm)

머리 속에 있는 알고리즘을 정확하고 빠르게 프로그램으로 작성하는 과정

문제를 해결하기 위한 일련의 절차나 방법을 공식화한 상태



01. 구현 알고리즘

Lv 1.별 찍기

```
    *
   ***
  *****
 *******
*********
 *******
  *****
   ***
    *
```

첫째줄에는 1개, 둘째줄에는 3개 세번째 줄에는 5개 이렇게 2개씩 증가하다가 5번째 줄에서 가장 많은 9개를 찍고 다시 2개씩 줄어 1개로 돌아오는 형태

01. 구현 알고리즘

Lv 1.별 찍기

```
   *
  ***
 *****
*******
*********
*******
 *****
  ***
   *
```


```
   *
  ***
 *****
*******
*********
-------
*******
 *****
  ***
   *
```

```
oooo*
ooo***
oo*****
o*******
*********
---------
o*******
oo*****
ooo***
oooo*
```

01. 구현 알고리즘

Lv 1.별 찍기

```
oooo*
ooo***
oo*****
o*******
*********
```

```
o*******
oo*****
ooo***
oooo*
```


```
N = int(input())
for i in range(0, N):
  print(" "*(N - i - 1) + "*"*(1 + i * 2))
for j in range(0, N - 1):
  print(" "*(j + 1) + "*"*(2 * N - 3 - (2 * j)))
```

01. 구현 알고리즘

Lv 2. 완전 탐색 – 문자열 압축

문자열에서 같은 값이 연속으로 나타나는 것을 그 문자의 개수와 반복되는 값으로 표현하여 더 짧은 문자열로 줄여서 표현하는 알고리즘

aabbacc -> 2a2ba3c
abcabcdede -> abcabcdede

문자열을 1개 이상의 단위로 잘라서 압축하여 더 짧은 문자열로 표현할 수 있는지

ababcdcdababcdcd -> 2ab2cd2ab2cd (2개 단위)
->2ababcdcd (8개 단위)

abcabcdede -> abcabc2de(2개 단위)
-> 2abcdede(3개 단위)

⇒ 압축할 문자열 s가 매개변수로 주어질 때, 1개 이상의 단위로 문자열을 잘라 압축하여 표현한 문자열 중 가장 짧은 것의 길이를 return 하도록 함

01. 구현 알고리즘

Lv 2. 완전 탐색 – 문자열 압축

a a a a b b a b b a b b

#case 1 (step = 1) => 4a2ba2ba2b
a a a a b b a b b a b b

#case 2 (step = 2) => 2aabbabbabb
a a a a b b a b b a b b

#case 3 (step = 3) => aaa3abb
a a a a b b a b b a b b

#case 4 (step = 4) => aaaabbabbabb
a a a a b b a b b a b b

#case 5 (step = 5) => aaaabbabbabb
a a a a b b a b b a b b

#case 6 (step = 6) => aaaabbabbabb
a a a a b b a b b a b b

01. 구현 알고리즘

Lv 2. 완전 탐색 – 문자열 압축

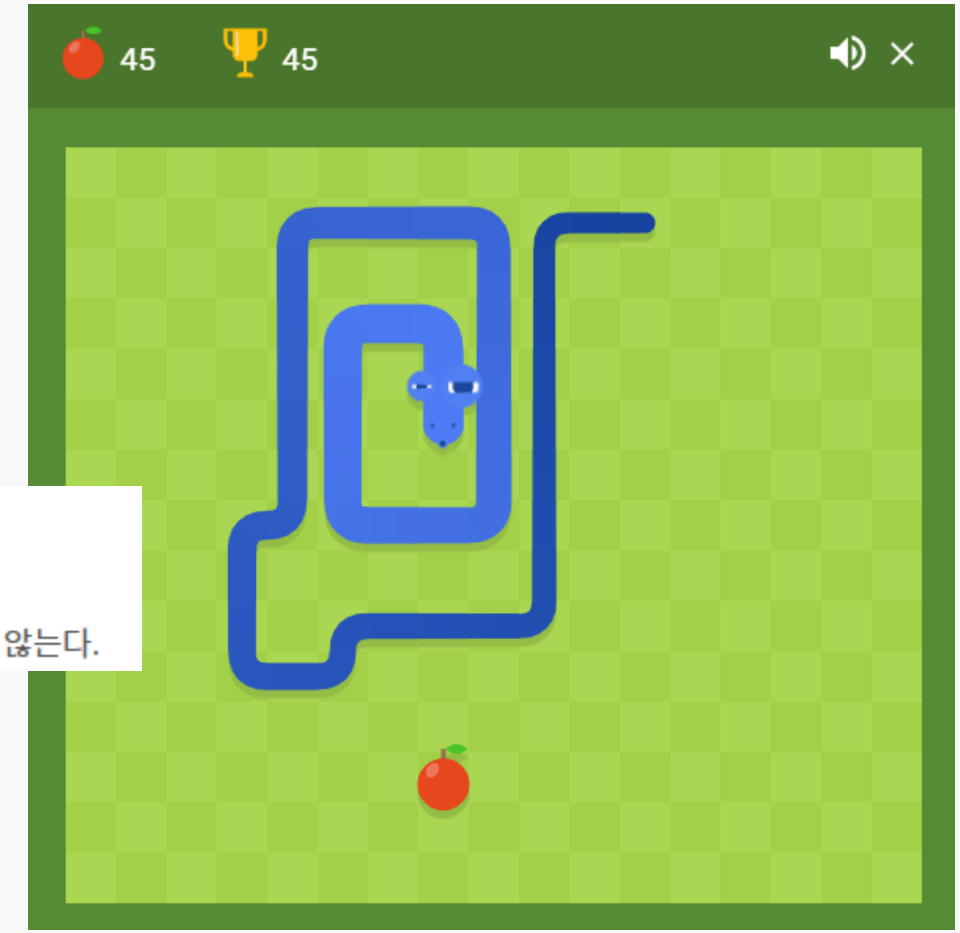
```
def solution(s):
    answer = len(s)
    for i in range(1, len(s)//2+1):
        compressed = ""
        prev=s[0:step]
        count=1
        for j in range(step,len(s),step):
            if prev == s[j:j+step]:
                count += 1
            else:
              compressed+=str(count)+prev if count >=2 else prev
              prev = s[j:j+step]
              count=1

        compressed +=str(count)+ prev if count >= 2 else prev
        answer = min(answer, len(compressed))
    return (answer)
```

01. 구현 알고리즘

Lv 3. 시뮬레이션

N x N 정사각 보드 위에서 뱀 게임을 하려고 합니다
이때 몇몇 칸에는 사과가 놓여있고
보드의 상하좌우 끝에는 벽이 있습니다
게임을 시작할 때 뱀은 맨 위 맨 좌측에 위치하고
뱀의 길이는 1
처음에 오른쪽으로 향합니다

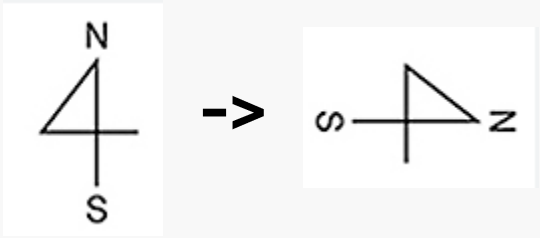


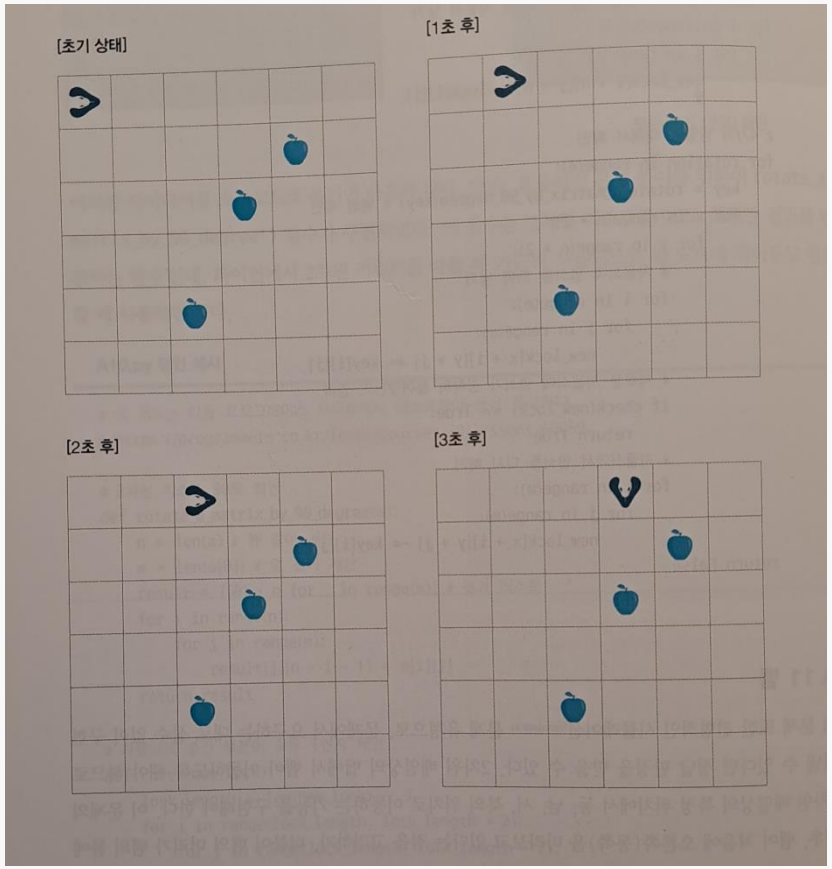
- 먼저 뱀은 몸길이를 늘려 머리를 다음칸에 위치시킨다.
- 만약 이동한 칸에 사과가 있다면, 그 칸에 있던 사과가 없어지고 꼬리는 움직이지 않는다.
- 만약 이동한 칸에 사과가 없다면, 몸길이를 줄여서 꼬리가 위치한 칸을 비워준다. 즉, 몸길이는 변하지 않는다.

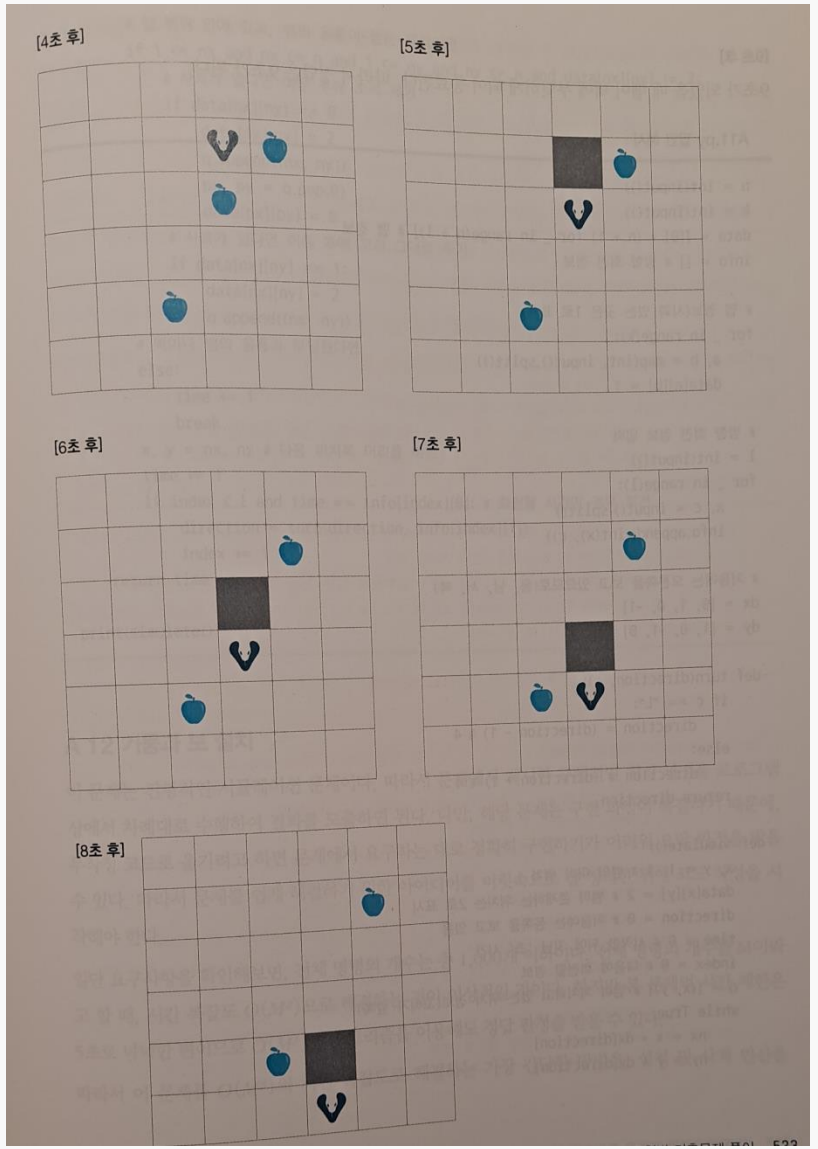
사과의 위치와 뱀의 이동 경로가 주어졌을 때,
이 게임은 몇 초에 끝나는지 계산하세요

01. 구현 알고리즘

Lv 3. 시뮬레이션







01. 구현 알고리즘

Lv 3. 시뮬레이션

```python
n = int(input())
k = int(input())
data = [[0] * (n + 1) for _ in range(n + 1)]
info = []

for _ in range(k):
    a, b = map(int, input().split())
    data[a][b] = 1

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
```

```python
def simulate():
    x, y = 1, 1
    data[x][y] = 2
    direction = 0
    time = 0
    index = 0
    q = [(x, y)]

    while True:
        nx = x + dx[direction]
        ny = y + dy[direction]
        if 1 <= nx and nx <= n and 1 <= ny and ny <= n and data[nx][ny] != 2:
            if data[nx][ny] == 0:
                data[nx][ny] = 2
                q.append((nx, ny))
                px, py = q.pop(0)
                data[px][py] = 0
            if data[nx][ny] == 1:
                data[nx][ny] = 2
                q.append((nx, ny))
        else:
            time += 1
            break
        x, y = nx, ny
        time += 1
        if index < l and time == info[index][0]:
            direction = turn(direction, info[index][1])
            index += 1
    return time

print(simulate())
```

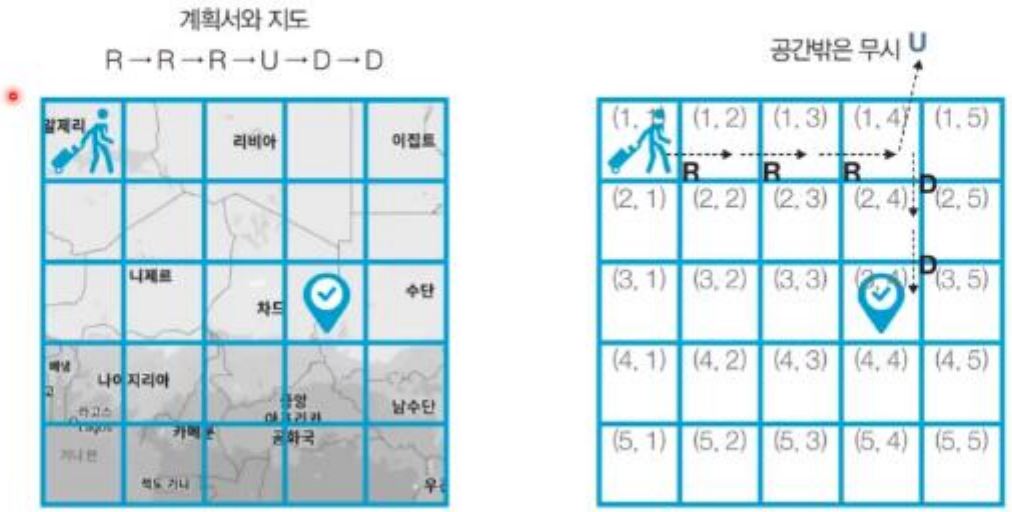
Quiz

***2개의 문제 중에서 본인이 풀 수 있는 문제를 골라서 풀이하세요 (둘 다 풀어도 상관없습니다)**

#1. 상하좌우 (난이도 ★ 출처. 이것이 코딩테스트다 파이썬편)

- 여행가 A는 **N × N** 크기의 정사각형 공간 위에 서 있습니다. 이 공간은 1 × 1 크기의 정사각형으로 나누어져 있습니다. 가장 왼쪽 위 좌표는 (1, 1)이며, 가장 오른쪽 아래 좌표는 (N, N)에 해당합니다. 여행가 A는 **상, 하, 좌, 우 방향으로 이동**할 수 있으며, 시작 좌표는 항상 **(1, 1)**입니다. 우리 앞에는 여행가 A가 이동할 계획이 적힌 계획서가 놓여 있습니다.
- 계획서에는 하나의 줄에 띄어쓰기를 기준으로 하여 L, R, U, D 중 하나의 문자가 반복적으로 적혀 있습니다. 각 문자의 의미는 다음과 같습니다.
  - L: 왼쪽으로 한 칸 이동
  - R: 오른쪽으로 한 칸 이동
  - U: 위로 한 칸 이동
  - D: 아래로 한 칸 이동

- 이때 여행가 A가 N × N 크기의 정사각형 공간을 벗어나는 움직임은 무시됩니다. 예를 들어 (1, 1)의 위치에서 L 혹은 U를 만나면 무시됩니다. 다음은 N = 5인 지도와 계획서입니다.



입력 조건
- 첫째 줄에 공간의 크기를 나타내는 N이 주어집니다. (1 <= N <= 100)
- 둘째 줄에 여행가 A가 이동할 계획서 내용이 주어집니다. (1 <= 이동 횟수 <= 100)

출력 조건
- 첫째 줄에 여행가 A가 최종적으로 도착할 지점의 좌표 (X, Y)를 공백을 기준으로 구분하여 출력합니다.

입력 예시

```
5
R R R U D D
```

출력 예시

```
3 4
```

Quiz

***다음은 시뮬레이션 유형의 문제입니다. 조건을 읽고 알맞은 코드를 작성하세요**

#2. 게임 개발 (난이도. ★★, 출처. 이것이 코딩테스트다 파이썬)

문제

현민이는 게임 캐릭터가 맵 안에서 움직이는 시스템을 개발 중이다. 캐릭터가 있는 장소는 1 X 1 크기의 정사각형으로 이뤄진 N X M 크기의 직사각형으로, 각각의 칸은 육지 또는 바다이다. 캐릭터는 동서남북 중 한 곳을 바라본다.

맵의 각 칸은 (A, B)로 나타낼 수 있고, A는 북쪽으로부터 떨어진 칸의 개수, B는 서쪽으로부터 떨어진 칸의 개수이다. 캐릭터는 상하좌우로 움직일 수 있고, 바다로 되어 있는 공간에는 갈 수 없다. 캐릭터의 움직임을 설정하기 위해 정해 놓은 매뉴얼은 이러하다.

현재 위치에서 현재 방향을 기준으로 왼쪽 방향(반시계 방향으로 90도 회전한 방향)부터 차례대로 갈 곳을 정한다.

캐릭터의 바로 왼쪽 방향에 아직 가보지 않은 칸이 존재한다면, 왼쪽 방향으로 회전한 다음 왼쪽으로 한 칸을 전진한다. 왼쪽 방향에 가보지 않은 칸이 없다면, 왼쪽 방향으로 회전만 수행하고 1단계로 돌아간다.

만약 네 방향 모두 이미 가본 칸이거나 바다로 되어 있는 칸인 경우에는, 바라보는 방향을 유지한 채로 한 칸 뒤로 가고 1단계로 돌아간다. 단, 이때 뒤쪽 방향이 바다인 칸이라 뒤로 갈 수 없는 경우에는 움직임을 멈춘다.

현민이는 위 과정을 반복적으로 수행하면서 캐릭터의 움직임에 이상이 있는지 테스트하려고 한다. 메뉴얼에 따라 캐릭터를 이동시킨 뒤에, 캐릭터가 방문한 칸의 수를 출력하는 프로그램을 만드시오.

입력
첫째 줄에 맵의 세로 크기 N과 가로 크기 M을 공백으로 구분하여 입력한다.
(3 <= N, M <= 50)
둘째 줄에 게임 캐릭터가 있는 칸의 좌표 (A, B)와 바라보는 방향 d가 각각 서로 공백으로 구분하여 주어진다. 방향 d의 값으로는 다음과 같이 4가지가 존재한다.
0 : 북쪽 1 : 동쪽 2 : 남쪽 3 : 서쪽
셋째 줄부터 맵이 육지인지 바다인지에 대한 정보가 주어진다. N개의 줄에 맵의 상태가 북쪽부터 남쪽 순서대로, 각 줄의 데이터는 서쪽부터 동쪽 순서대로 주어진다. 맵의 외각은 항상 바다로 되어 있다.
0 : 육지
1 : 바다
처음에 게임 캐릭터가 위치한 칸의 상태는 항상 육지이다.
출력
첫째 줄에 이동을 마친 후 캐릭터가 방문한 칸의 수를 출력한다.
입력 예시
4 4
1 1 0 // (1, 1)에 북쪽(0)을 바라보고 서 있는 캐릭터
1 1 1 1
1 0 0 1
1 1 0 1
1 1 1 1
출력 예시
3