구현 - 4장

구현이란?

코딩 테스트에서 구현은 **머리 속에 있는 알고리즘을 소스 코드로 바꾸는 과정**이다. 어떤 문제를 풀든 코드 작성은 필수이기 때문에 모든 문제를 구현의 범위에 포함 시킬 수도 있다.

흔히 문제 해결 분야에서 구현 유형의 문제는 **풀이를 떠올리기는 쉽지만 코드로 옮기기 어려운 문제**를 의미한다. 개발할 때 프로그래밍 언어의 문법에 능숙하고 코드 작성 속도(타자) 가 빠른 사람을 보고 '피지컬이 좋다' 라고 얘기하는데, 구현 문제는 그런 의미에서 피지컬을 요구하는 문제라고 볼 수도 있다.

이취코에서는 **완전 탐색, 시뮬레이션** 유형을 구현 유형으로 묶어 분류한다.

완전 탐색이란 **모든 경우의 수를 주저 없이 다 계산**하는 해결 방법을 의미하고, 시뮬레이션은 **문제에서 제시한 알고리즘을 한 단계씩 차례대로 직접 수행**해야 하는 문제 유형을 의미한다.

구현 시 고려해야 할 메모리 제약 사항

파이썬에서 여러 개의 변수를 사용할 때는 리스트를 이용한다. 하지만 코딩 테스트에서는 메모리 제한에 대해 고려해야 한다. 대체로 코딩 테스트에서는 128~512 MB로 메모리를 제한하는데 알고리즘 문제 중 때로는 수백만 개 이상의 데이터를 처리해야 하는 경우가 출제되곤 한다.

파이썬에선 int 자료형을 별도로 명시해 줄 필요는 없지만 시스템 적으로는 정수 데이터가 1000만개 정도 있으면 약 40MB의 메모리를 사용한다. 즉, 리스트를 여러 개 선언하고, 그 중에서 크기가 1000만 이상인 리스트가 있다면 메모리 용량 제한으로 문제를 풀 수 없게 되는 경우도 있다. 물론, 1000만개 이상의 데이터는 입출력에 너무 많은 시간이 소요되며 채점 중 다양한 문제가 발생할 수 있기에 잘 나오지는 않는다.

따라서 일반적인 코딩 테스트 수준에서는 메모리 사용량 제한보다 더 적은 크기의 메모리를 사용해야 한다는 점 정도만 기억하면 된다.

또한 일반적인 기업 코딩 테스트 환경에서는 파이썬으로 제출한 코드가 1초에 2000만 번의 연산 정도를 수행한다. 그렇기 때문에 시간 제한이 1초이고, 데이터의 개수가 100만 개인 문제가 있다면 O(NlogN) 안에 풀어야 하는 문제라고 생각하면 된다. 왜냐하면 N=100만일 때, NlogN이 약 2000만이기 때문이다.

구현 문제 접근법

보통 구현 문제는 사소한 입력 조건 등을 문제에서 명시해주며 문제의 길이가 꽤 길다. 하지만 고차원적인 사고력을 요구하는 문제는 나오지 않는 편이라 문법에 익숙하다면 오히려 쉽게 풀 수 있다.

예제 1. 상하좌우

💡 문제

여행가 A는 N X N 크기의 정사각형 공간 위에 서 있다. 이 공간은 1 X 1 크기의 정사각형으로 나누어져 있다. 가장 왼쪽 위 좌표는 (1, 1)이며, 가장 오른쪽 아래 좌표는 (N, N)에 해당한다. 여행가 A는 상, 하, 좌, 우 방향으로 이동할 수 있으며, 시작 좌표는 항상 (1, 1) 이다. 우리 앞에는 여행가 A가 이동할 계획이 적힌 계획서가 놓여 있다.

계획서에는 하나의 줄에 띄어쓰기를 기준으로 하여 L, R, U, D 중 하나의 문자가 반복적으로 적혀 있다. 예를 들어 R R R U D D와 같이 나와있으며, 여행가 A가 N X N 크기의 정사각형 공간을 벗어나는 움직임은 무시된다.

이때, 여행가가 최종적으로 도착할 목적지의 좌표를 출력하라.

💡 해결

딱 문제에 나온 대로 좌표 이동만 시켜주면 된다.

💡 코드

```
# 내 정답
N = int(input())
paths = list(map(str, input().split()))

x = 1
y = 1

for path in paths:
    if path == 'L':
        if y - 1 >= 1:
            y -= 1
    elif path == 'R':
        if y + 1 <= N:
            y += 1
    elif path == 'U':
        if x - 1 >= 1:
            x -= 1
    else:
        if x + 1 <= N:
            x += 1

print(x, y)

# 답안
n = int(input())
x, y = 1, 1
plans = input().split()

dx = [0, 0, -1, 1]
dy = [-1, 1, 0, 0]
move_types = ['L', 'R', 'U', 'D']

for plan in plans:
  for i in range(len(move_types)):
    if plan == move_types[i]:
      nx = x + dx[i]
      ny = y + dy[i]

  if nx < 1 or ny < 1 or nx > n or ny > n:
    continue
  x, y = nx, ny

print(x, y)
```

예제 2. 시각

💡 문제

정수 N이 입력되면 00시 00분 00초부터 N시 59분 59초까지의 모든 시각 중에서 3이 하나라도 포함되는 모든 경우의 수를 구하는 프로그램을 작성하시오. 예를 들어 1을 입력했을 때, 다음은 3이 하나라도 포함되어 있으므로 세어야 하는 시각이다.

- 00시 00분 03초
- 00시 13분 30초

💡 해결

모든 경우의 수는 총 N*60*60 개 이며, 시간 제한은 2초이기에 꽤 널널한 편이다. 모든 경우에 대해 해결하면 된다.

💡 코드

```
# 내 정답
N = int(input())

cnt = 0

for i in range((N + 1) * 60 * 60):
    s = str(i % 60)
    m = str(i % 3600 // 60)
    h = str(i // 3600)

    if '3' in h + m + s:
        cnt += 1

print(cnt)

# 답안
h = int(input())

cnt = 0
```

```
for i in range(h + 1):
  for j in range(60):
    for k in range(60):
      if '3' in str(i) + str(j) + str(k):
        cnt += 1

print(cnt)
```

예제 3. 왕실의 나이트

💡 문제

행복 왕국의 왕실 정원은 체스판과 같은 8 X 8 좌표 평면이다. 왕실 정원의 특정 한 칸에 나이트가 서 있다.

나이트는 항상 L자 형태로만 이동할 수 있어, 다음 2가지 경우로만 이동할 수 있다.

- 수평으로 2칸 이동 후, 수직으로 한 칸 이동
- 수직으로 2칸 이동 후, 수평으로 1칸 이동

8 X 8 좌표 평면상에서 나이트의 위치가 주어질 때, 나이트가 이동할 수 있는 경우의 수를 출력하라.

💡 해결

한 번의 움직임만 보면 되기 때문에 총 8가지의 경우의 수에 대해 모두 확인하면 된다. 앞의 예제 1. 상하좌우와도 완전히 동일하다.

💡 코드

```
# 내 정답
data = input()

x = data[0]
y = int(data[1])

x_path = ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h']
plans = [(2, 1), (2, -1), (1, 2), (1, -2), (-1, 2), (-1, -2), (-2, 1), (-2, -1)]

cnt = 0

for plan in plans:
    nx = x_path.index(x) + plan[0]
    ny = y + plan[1]

    if nx >= 0 and ny >= 1 and nx <= 7 and ny <= 8:
        cnt += 1

print(cnt)
```

예제 4. 게임 개발

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 메뉴얼은 아래와 같다.

1. 현재 위치에서 현재 방향을 기준으로 왼쪽 방향(반시계 방향으로 90도 회전한 방향)부터 차례대로 갈 곳을 정한다.
2. 캐릭터의 바로 왼쪽 방향에 아직 가보지 않은 칸이 존재한다면, 왼쪽 방향으로 회전 후 왼쪽으로 한 칸을 전진한다. 왼쪽 방향에 가보지 않은 칸이 없다면, 왼쪽 방향으로 회전 하고 1단계로 돌아간다.

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

💡 해결

상하좌우 문제와 유사하게 기본 변수 세팅을 하고 문제에 제시된 대로 코드를 작성했다.

하지만 조건 3번이 이해가 가지 않는다. 코드를 작성하다보면 필연적으로 방문한 육지는 1로 데이터를 바꿔 바다와 동일하게 해준다. 이 상태에서 조건 3번이... 네 방향 모두 갈 수 없는 곳인데 뒤로 한 칸 이동한다는게 무슨 말인지.. 아래 코드로도 답은 동일하게 나온다.

조건 3번에 대한 추가 테스트 케이스가 명시되어 있으면 좋겠다..

💡 코드

```
# 내 코드
n, m = map(int, input().split())
x, y, dir = map(int, input().split())

maps = []

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

for i in range(n):
    data = list(map(int, input().split()))
    maps.append(data)

cnt = 1
turn_cnt = 0

while True:
    dir -= 1 # 회전
    turn_cnt += 1 # 회전 횟수 증가

    if dir < 0: # 0에서 회전하면 3으로
        dir = 3
```

```
    # 앞으로 전진
    nx = x + dx[dir]
    ny = y + dy[dir]

    # 이동한 좌표가 전체 직사각형 위에 있고
    if nx >= 0 and ny >= 0 and nx < n - 1 and ny < n - 1:
        # 이동한 곳이 육지면
        if maps[nx][ny] == 0:
            # 방문하고
            maps[nx][ny] = 1
            # 좌표도 바꾸고
            x = nx
            y = ny
            # 방문한 땅 횟수 증가
            cnt += 1
            # 회전 횟수 초기화
            turn_cnt = 0

    # 전 방향 다 돌았는데 갈 곳이 없으면 종료
    if turn_cnt == 3:
        break

print(cnt)
```