(2) 구현

‘ 머릿속에 있는 알고리즘을 소스코드로 바꾸는 과정’

→ 결국 모든 범위의 코딩테스트 문제 유형을 포함하는 개념!

 풀이를 떠올리는 것은 쉽지만 소스코드로 옮기기 어려운 문제! 피지컬이 중요!

ex)

완전 탐색	: 모든 경우의 수를 주저 없이 다 계산하는 해결 방법
시뮬레이션	:문제에서 제시한 알고리즘을 한단계씩 차례대로 직접 수행해야 하는 문제

구현 시 고려해야 할 메모리 제약 사항

- 파이썬에서는 프로그래머가 직접 자료형을 지정할 필요X
- 매우 큰 수의 연산 또한 기본으로 지원
- but, 실수형 변수는 다른 언어와 마찬가지로 유효 숫자에 따라 연산 결과가 원하는 값이 나오지 않을 수 있음!

파이썬에서 리스트 크기

- 리스트 이용시 코딩테스트의 메모리 제한을 고려해야 함!
- int 자료형 데이터의 개수에 따른 메모리 사용량

데이터의 개수 (리스트의 길이)	메모리 사용량
1,000	약 4KB
1,000,000	약 4MB
10,000,000	약 40MB

채점 환경

일반적으로 시간 제한이 1초이고 데이터의 개수가 100만 개인 문제

→ 시간복잡도 O(NlogN)이내의 알고리즘 이용해야 함!

- 시간 제한: 1초
- 메모리 제한: 128MB

구현 문제에 접근하는 방법

In 파이썬, 구현 난이도는 쉬운 편, but, 프로그램 실행 시간은 길다!

파이썬에선 API 개발 문제 접근 easy!

예제 4-1. 상하좌우

- N X N 크기의 정사각형 공간, 상하좌우 움직일 수 있음! but, 공간을 벗어나는 움직임은 무시됨!
- 최종적으로 도착할 지점의 좌표 (X,Y)를 출력하면 됨!
- 일련의 명령에 따라서 개체를 차례대로 이동시키는 **시뮬레이션 유형**!

```
n = int(input()) # 공간의 크기 N 입력
x,y = 1,1 # 시작점
plans = input().split() # 이동 경로 입력

# L,R,U,D 에 따른 이동 방향
dx = [0,0,-1,1] # L,R,U,D x좌표
dy = [-1,1,0,0] # L,R,U,D y좌표
move_types = ['L','R','U','D']

# 이동 계획을 하나씩 확인
for plan in plans:
  # 이동 후 좌표 구하기
  for i in range(len(move_types)):
    if plan == move_types[i]:
      nx = x + dx[i]
      ny = y + dy[i]
  # 공간을 벗어날때는 무시!
  if nx < 1 or ny < 1 or nx > n or ny > n:
    continue

# 이동 수행 - 대입
x, y = nx, ny

print(x, y)
```

→ 연산 횟수는 이동횟수에 비례

→ 시간 복잡도 : O(N)

예제 4-2 시각

- 정수 N이 입력되면 00시 00분 00초 부터 N시 59분 59초까지의 모든 시각 중 3이 하나라도 포함되는 모든 경우의 수 구하기
- 전체 경우의 수 86,400가지! → 문자열 연산을 이용해도 시간 제한 2초안에 해결 가능!
- **완전 탐색 유형 :** 가능한 경우의 수를 모두 검사해보는 방법! → 데이터 개수가 큰 경우엔 bad
- 경우의 수가 100만개 이하일때 사용하면 적절!

2. 실전 문제 - 왕실의 나이트

- 8 X 8 좌표 평면, 나이트는 L자 형태로만 이동할 수 있고, 정원 밖으론 못 나감! 2 가지 경우로 행동

1. **수평으로 두 칸 이동한 뒤에 수직으로 한 칸 이동하기**
2. **수직으로 두 칸 이동한 뒤에 수평으로 한 칸 이동하기**

나이트의 위치가 주어졌을 때, 나이트가 이동할 수 있는 경우의 수를 출력!

행위치: 1~8 / 열 위치 : a ~ h

3. 실전 문제 - 게임 개발

- N X M 크기의 직사각형
- 맵의 각 칸은 (A,B) , A: 북쪽으로부터 떨어진 칸의 개수, B: 서쪽으로부터 떨어진 칸의 개수
- 바다로 된 곳에는 갈 수 없음. 캐릭터는 상하좌우로 움직임

1. **현재 위치에서 현재 방향을 기준으로 왼쪽 방향부터 차례대로 갈 곳을 정한다**
2. **캐릭터의 바로 왼쪽 방향에 아직 가보지 않은 칸이 존재한다면, 왼쪽 방향으로 회전한 다음 왼쪽으로 한칸을 전진한다. 왼쪽 방향에 가보지 않은 칸이 없다면, 왼쪽 방향으로 회전만 수행하고 1단계로 돌아간다.**

3. **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.**

→ dx, dy라는 별도의 리스트를 만들어서 방향을 정하기!

→ 리스트 컴프리헨션 문법을 사용해 2차원 리스트 초기화