아래 강의 영상에서 배운 내용을 정리해보려고 한다. 가감된 부분이 있을 수 있다.
https://youtu.be/2zjoKjt97vQ?list=PLRx0vPvlEmdAghTr5mXQxGpHjWqSz0dgC

그리디 알고리즘

탐욕법이라고도 불린다.
현재 상황에서 지금 당장 좋은 것만 고르는 방법을 의미한다.
문제를 풀기 위한 최소한의 아이디어를 떠올릴 수 있는 능력을 요구한다.
그리디 해법은 그 정당성 분석이 중요하다.
단순히 가장 좋아 보이는 것을 반복적으로 선택해도 최적의 해를 구할 수 있는지 검토해보고,
그렇다면 탐욕법을 써서 문제를 풀면 된다.

탐욕법은 최적의 해를 보장할 수 없을 때가 많다.
하지만 코딩 테스트에서 대부분의 그리디 문제는 **탐욕법으로 얻은 해가 최적의 해가 되는 상황에서,**
이를 추론할 수 있어야 풀리도록 출제된다.

탐욕법 예시: 거스름 돈 문제

가장 큰 화폐 단위부터 돈을 거슬러 주는 것이 최적의 해를 보장하는 이유는
가지고 있는 동전 중에서 **큰 단위가 항상 작은 단위의 배수이므로 작은 단위의 동전들을 종합해 다른 해가 나올 수 없기 때문**이다.
화폐의 종류가 K라고 할 때, 소스코드의 시간 복잡도는 O(K)이다.
아래는 https://www.acmicpc.net/problem/5585에 제출한 답안들이다.
강의 듣기 전 푼 답안

```
import sys

data = sys.stdin.readline().rstrip()
m = 1000 - int(data)
n = m // 500 + m % 500 // 100 + m % 100 // 50 + m % 50 // 10 + m % 10 // 5 + m % 5

print(n)
```

강의 들은 후 푼 답안
역시 반복되는 건 반복문으로 처리하는 게 깔끔하다. 갈 길이 멀다.

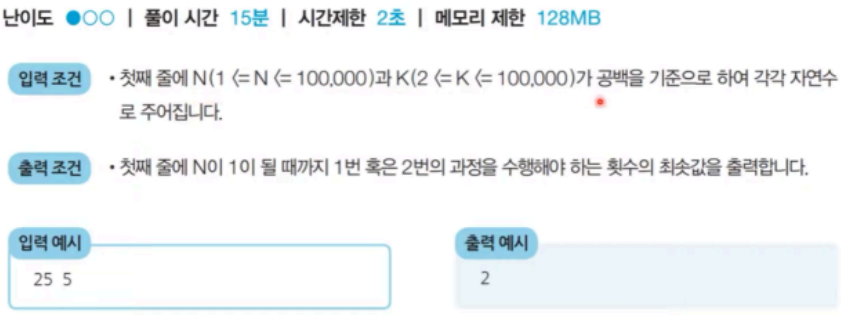
```python
import sys

data = sys.stdin.readline().rstrip()
n = 1000 - int(data)
count = 0

# 큰 단위의 화폐부터 차례대로 확인하기
array = [500, 100, 50, 10, 5, 1]

for coin in array:
    count += n // coin # 해당 화폐로 거슬러 줄 수 있는 동전의 개수 세기
    n %= coin

print(count)
```

탐욕법 예시: 1이 될 때까지

〈문제〉 1이 될 때까지: 문제 설명

- 어떠한 수 **N이 1이 될 때까지** 다음의 두 과정 중 하나를 반복적으로 선택하여 수행하려고 합니다. 단, 두 번째 연산은 N이 K로 나누어 떨어질 때만 선택할 수 있습니다.
  1. N에서 1을 뺍니다.
  2. N을 K로 나눕니다.
- 예를 들어 N이 17, K가 4라고 가정합시다. 이때 1번의 과정을 한 번 수행하면 N은 16이 됩니다. 이후에 2번의 과정을 두 번 수행하면 N은 1이 됩니다. 결과적으로 이 경우 전체 과정을 실행한 횟수는 3이 됩니다. 이는 N을 1로 만드는 최소 횟수입니다.
- N과 K가 주어질 때 N이 1이 될 때까지 1번 혹은 2번의 **과정을 수행해야 하는 최소 횟수**를 구하는 프로그램을 작성하세요.

〈문제〉 1이 될 때까지: 문제 조건



N의 값을 줄일 때 2 이상의 수로 나누는 작업이 1을 빼는 작업보다 수를 훨씬 많이 줄일 수 있기 때문에
주어진 N에 대하여 최대한 많이 나누기를 수행하면 된다.
탐욕법을 썼을 때 최적의 해를 구할 수 있는지 정당성 분석을 하자면
나누는 수인 K가 2 이상이면 K로 나누는 것이 1을 빼는 것보다 항상 빠르게 N을 줄일 수 있고, N은 항상 1에 도달할 수 있으므로 OK다.

백준에서 비슷한 문제를 봤던 것 같은데 못 찾았다.
처음에 답안 보고 좀 어질어질했는데 천천히 소스 코드를 읽어봤더니 이해가 됐다.

```python
# N, K를 공백을 기준으로 구분하여 입력 받기
n, k = map(int, input().split())

result = 0

while True:
    # N이 K로 나누어 떨어지는 수가 될 때까지 빼기
    target = (n // k) * k
    result += (n - target)
    n = target
    # N이 K보다 작을 때 (더 이상 나눌 수 없을 때) 반복문 탈출
    if n < k:
      break
    # K로 나누기
    result += 1
    n //= k

# 마지막으로 남은 수에 대하여 1씩 빼기
result += (n - 1)
print(result)
```

탐욕법 예시: 곱하기 혹은 더하기

〈문제〉 곱하기 혹은 더하기: 문제 설명

- 각 자리가 숫자(0부터 9)로만 이루어진 문자열 S가 주어졌을 때, 왼쪽부터 오른쪽으로 하나씩 모든 숫자를 확인하며 숫자 사이에 '×' 혹은 '+' 연산자를 넣어 결과적으로 **만들어질 수 있는 가장 큰 수를 구하는 프로그램**을 작성하세요. 단, +보다 ×를 먼저 계산하는 일반적인 방식과는 달리, **모든 연산은 왼쪽에서부터 순서대로** 이루어진다고 가정합니다.
- 예를 들어 02984라는 문자열로 만들 수 있는 가장 큰 수는 ((((0 + 2) × 9) × 8) × 4) = 576입니다. 또한 만들어질 수 있는 가장 큰 수는 항상 20억 이하의 정수가 되도록 입력이 주어집니다.

〈문제〉 곱하기 혹은 더하기: 문제 조건

난이도 ●○○ | 풀이 시간 30분 | 시간 제한 1초 | 메모리 제한 128MB | 기출 Facebook 인터뷰

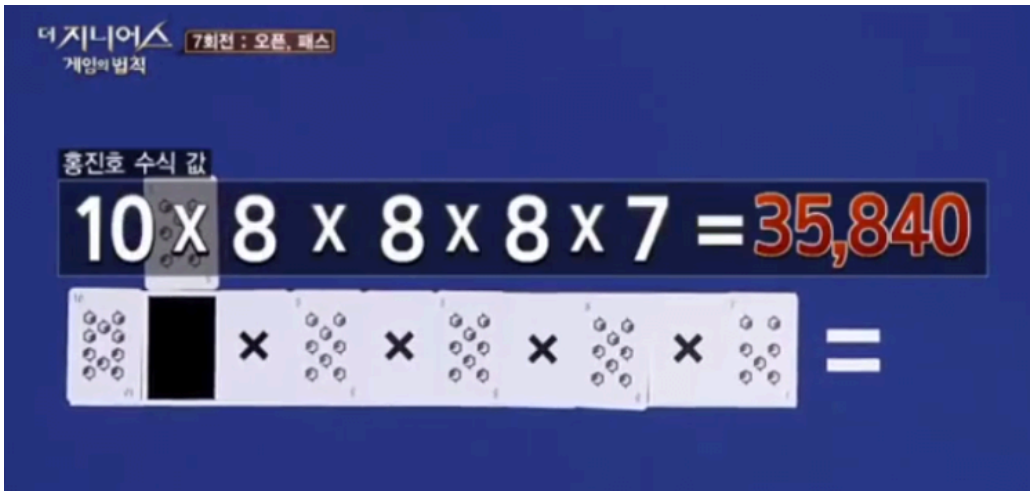
입력 조건 • 첫째 줄에 여러 개의 숫자로 구성된 하나의 문자열 S가 주어집니다. (1 ≤ S의 길이 ≤ 20)

출력 조건 • 첫째 줄에 만들어질 수 있는 가장 큰 수를 출력합니다.

입력 예시 1	출력 예시 1
02984	576

입력 예시 2	출력 예시 2
567	210

지니어스 본 사람들은 알 만한데, 오픈 패스가 생각나는 문제다. 의도하셨나?



추억의 콩픈패스

만들어지는 수가 최대 20억 이하로 문자열이 주어진다는 건 int 범위를 고려한 것으로 보인다.
앞서 빨셈보다 나눗셈이 더 값을 작게 만들었던 것처럼, 덧셈보다 곱셈이 더 값을 크게 만든다는 점에서 문제 해결 아이디어를 얻으면 된다.
다만 두 수 중에서 하나라도 0 혹은 1인 경우, 곱하기보다는 더하기를 수행하는 것이 효율적이다.
따라서 두 수에 대하여 연산을 수행할 때, **두 수 중에서 하나라도 1 이하인 경우에는 더하며, 두 수가 모두 2 이상인 경우에는 곱하면 정답**이다.

```
data = input()

# 첫 번째 문자를 숫자로 변경하여 대입
result = int(data[0])

for i in range(1, len(data)):
    # 두 수 중에서 하나라도 0 혹은 1인 경우, 곱하기보다는 더하기 수행
    num = int(data[i])
    if num <= 1 or result <= 1:
        result += num
    else:
        result *= num

print(result)
```

탐욕법 예시: 모험가 길드

〈문제〉 모험가 길드: 문제 설명

- 한 마을에 모험가가 N명 있습니다. 모험가 길드에서는 N명의 모험가를 대상으로 '공포도'를 측정했는데, '공포도'가 높은 모험가는 쉽게 공포를 느껴 위험 상황에서 제대로 대처할 능력이 떨어집니다.
- 모험가 길드장인 동빈이는 모험가 그룹을 안전하게 구성하고자 공포도가 X인 모험가는 반드시 X명 이상으로 구성한 모험가 그룹에 참여해야 여행을 떠날 수 있도록 규정했습니다.
- 동빈이는 최대 몇 개의 모험가 그룹을 만들 수 있는지 궁금합니다. N명의 모험가에 대한 정보가 주어졌을 때, **여행을 떠날 수 있는 그룹 수의 최댓값**을 구하는 프로그램을 작성하세요.

〈문제〉 모험가 길드: 문제 설명

- 예를 들어 N = 5이고, 각 모험가의 공포도가 다음과 같다고 가정합시다.

```
2 3 1 2 2
```

- 이 경우 그룹 1에 공포도가 1, 2, 3인 모험가를 한 명씩 넣고, 그룹 2에 공포도가 2인 남은 두 명을 넣게 되면 총 2개의 그룹을 만들 수 있습니다.
- 또한 몇 명의 모험가는 마을에 그대로 남아 있어도 되기 때문에, 모든 모험가를 특정한 그룹에 넣을 필요는 없습니다.

〈문제〉 모험가 길드: 문제 조건

난이도 ●○○ | 풀이 시간 30분 | 시간 제한 1초 | 메모리 제한 128MB | 기출 핵심 유형

입력 조건
- 첫째 줄에 모험가의 수 N이 주어집니다. (1 ≤ N ≤ 100,000)
- 둘째 줄에 각 모험가의 공포도의 값을 N 이하의 자연수로 주어지며, 각 자연수는 공백으로 구분합니다.

출력 조건
- 여행을 떠날 수 있는 그룹 수의 최댓값을 출력합니다.

입력 예시

```
5
2 3 1 2 2
```

출력 예시

```
2
```

인터넷을 찾아보니 모여라 게임으로 나오는데, OT나 MT 갔을 때 사회자가 몇 명 모여라 하면 뭉치는 그 게임 생각나는 문제다.
각자 최대 뭉칠 수 있는 인원수가 다르니 약간 다르긴 하다.
공포도로 오름차순 정렬한 후에, 앞에서부터 하나씩 확인하며 현재 그룹에 포함된 모험가의 수가 현재 확인하고 있는 공포도보다 크거나 같다면 이를 그룹으로 설정하면 된다. 두 번째 그림 예시에 이 방법을 적용한다면 1과 2, 2가 한 그룹이 되겠다.
소스 코드를 이해하는 것 자체는 쉬운데, 문제 해결 아이디어를 내고 정당성 분석을 하는 게 상당히 어려운 문제로 보인다.

```python
n = int(input())
data = list(map(int, input().split()))
data.sort()

result = 0 # 총 그룹의 수
count = 0 # 현재 그룹에 포함된 모험가의 수

for i in data: # 공포도를 낮은 것부터 하나씩 확인하며
    count += 1 # 현재 그룹에 해당 모험가를 포함시키기
    if count >= i: # 현재 그룹에 포함된 모험가의 수가 현재의 공포도 이상이라면, 그룹 결성
        result += 1 # 총 그룹의 수 증가시키기
        count = 0 # 현재 그룹에 포함된 모험가의 수 초기화

print(result) # 총 그룹의 수 출력
```

구현

머릿속에 있는 알고리즘을 소스코드로 바꾸는 과정이다.

알고리즘 대회에서 구현 유형의 문제란 **풀이를 떠올리는 것은 쉽지만 소스코드로 옮기기 어려운 문제**를 지칭한다.
구현 유형의 예시는 다음과 같다.

- 알고리즘은 간단한데 코드가 지나칠 만큼 길어지는 문제
- 실수 연산을 다루고, 특정 소수점 자리까지 출력해야 하는 문제
- 문자열을 특정한 기준에 따라서 끊어 처리해야 하는 문제
- 적절한 라이브러리를 찾아서 사용해야 하는 문제

일반적으로 알고리즘 문제에서의 이차원 공간은 행렬(Matrix)의 의미로 사용된다.
시뮬레이션 및 완전 탐색 문제에서는 2차원 공간에서의 **방향 벡터**가 자주 활용된다.

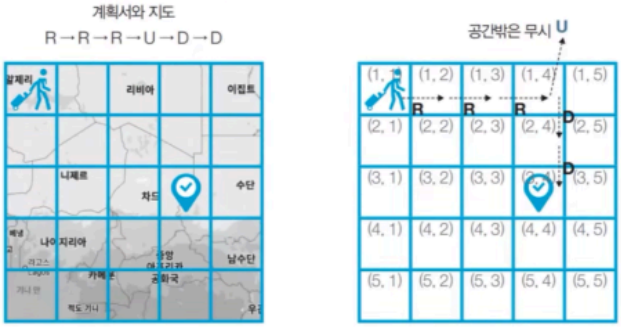
구현 예시: 상하좌우

〈문제〉 상하좌우: 문제 설명

- 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

〈문제〉 상하좌우: 문제 설명

- 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.



〈문제〉 상하좌우: 문제 조건

난이도 ●○○ | **풀이 시간** 15분 | **시간제한** 2초 | **메모리 제한** 128MB

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
```

요구사항대로 충실히 구현하면 되는 문제이다.

일련의 명령에 따라서 개체를 차례대로 이동시킨다는 점에서 **시뮬레이션(Simulation) 유형**으로도 분류되는 대표적인 구현 문제 유형이다.
다만 교재, 사이트에 따라 다르게 부르기도 하니 코딩 테스트에서의 시뮬레이션 유형, 구현 유형, 완전 탐색 유형은 서로 유사한 점이 많다는 정도로만 기억하면 된다.

```
# N 입력 받기
n = int(input())
x, y = 1, 1
plans = input().split()

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

# 이동 계획을 하나씩 확인하기
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 벗어나는 경우 무시
    if nx < 1 or ny < 1 or nx > n or ny > n:
        continue
    # 이동 수행
    x, y = nx, ny

print(x, y)
```

구현 예시: 시각

〈문제〉 시각: 문제 설명

- 정수 N이 입력되면 00시 00분 00초부터 N시 59분 59초까지의 **모든 시각 중에서 3이 하나라도 포함되는 모든 경우의 수를 구하는 프로그램을 작성**하세요. 예를 들어 1을 입력했을 때 다음은 3이 하나라도 포함되어 있으므로 세어야 하는 시각입니다.
  - 00시 00분 03초
  - 00시 13분 30초
- 반면에 다음은 3이 하나도 포함되어 있지 않으므로 세면 안 되는 시각입니다.
  - 00시 02분 55초
  - 01시 27분 45초

〈문제〉 시각: 문제 조건

난이도 ●○○ | 풀이 시간 15분 | 시간제한 2초 | 메모리 제한 128MB

입력 조건 • 첫째 줄에 정수 N이 입력됩니다. (0 <= N <= 23)

출력 조건 • 00시 00분 00초부터 N시 59분 59초까지의 모든 시각 중에서 3이 하나라도 포함되는 모든 경우의 수를 출력합니다.

입력 예시

```
5
```

출력 예시

```
11475
```

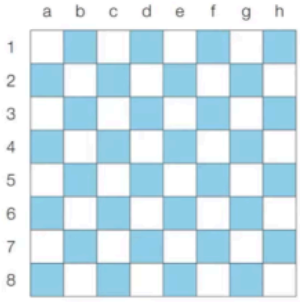
이 문제는 **가능한 모든 시각의 경우를 하나씩 모두 세서 풀 수 있는 문제**이다.
하루는 24 * 60 * 60 = 86400초이므로, 00시 00분 00초부터 23시 59분 59초까지의 모든 경우는 86400가지이다.
따라서 단순히 시각을 1씩 증가시키면서 3이 하나라도 포함되어 있는지를 확인하면 된다.
이처럼 가능한 경우의 수를 모두 검사해보는 탐색 방법을 사용하는 문제 유형을 **완전 탐색**(Brute Forcing) 문제 유형이라고 한다.

```
# H 입력 받기
h = int(input())

count = 0
for i in range(h + 1):
    for j in range(60):
        for k in range(60):
            # 매 시각 안에 '3'이 포함되어 있다면 카운트 증가
            if '3' in str(i) + str(j) + str(k):
                count += 1

print(count)
```

구현 예시: 왕실의 나이트

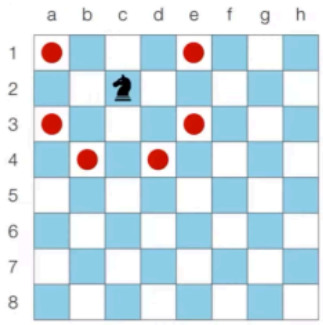
〈문제〉 왕실의 나이트: 문제 설명

- 행복 왕국의 왕실 정원은 체스판과 같은 **8 × 8 좌표 평면**입니다. 왕실 정원의 특정한 한 칸에 나이트가 서 있습니다. 나이트는 매우 충성스러운 신하로서 매일 무술을 연마합니다.
- 나이트는 말을 타고 있기 때문에 이동을 할 때는 L자 형태로만 이동할 수 있으며 정원 밖으로는 나갈 수 없습니다.
- 나이트는 특정 위치에서 다음과 같은 2가지 경우로 이동할 수 있습니다.
  1. 수평으로 두 칸 이동한 뒤에 수직으로 한 칸 이동하기
  2. 수직으로 두 칸 이동한 뒤에 수평으로 한 칸 이동하기



〈문제〉 왕실의 나이트: 문제 설명

- 이처럼 8 × 8 좌표 평면상에서 나이트의 위치가 주어졌을 때 나이트가 이동할 수 있는 경우의 수를 출력하는 프로그램을 작성하세요. 왕실의 정원에서 행 위치를 표현할 때는 1부터 8로 표현하며, 열 위치를 표현할 때는 a부터 h로 표현합니다.
  - c2에 있을 때 이동할 수 있는 경우의 수는 **6가지**입니다.



〈문제〉 왕실의 나이트: 문제 조건

난이도 ●○○ | **풀이 시간** 20분 | **시간 제한** 1초 | **메모리 제한** 128MB

입력 조건 • 첫째 줄에 8 × 8 좌표 평면상에서 현재 나이트가 위치한 곳의 좌표를 나타내는 두 문자로 구성된 문자열이 입력된다. 입력 문자는 a1처럼 열과 행으로 이뤄진다.

출력 조건 • 첫째 줄에 나이트가 이동할 수 있는 경우의 수를 출력하시오.

입력 예시

```
a1
```

출력 예시

```
2
```

요구사항대로 충실히 구현하면 되는 문제이다.
리스트를 이용하여 8가지 방향에 대한 방향 벡터를 정의하고,
나이트의 8가지 경로를 하나씩 확인하며 각 위치로 이동이 가능한지 확인하면 될 것이다.
여기서 ord()는 특정 문자를 아스키 값으로 변환하는 함수이다.

```
# 현재 나이트의 위치 입력받기
input_data = input()
row = int(input_data[1])
column = int(ord(input_data[0])) - int(ord('a')) + 1

# 나이트가 이동할 수 있는 8가지 방향 정의
steps = [(-2, -1), (-1, -2), (1, -2), (2, -1), (2, 1), (1, 2), (-1, 2), (-2, 1)]

# 8가지 방향에 대하여 각 위치로 이동이 가능한지 확인
result = 0
for step in steps:
    # 이동하고자 하는 위치 확인
    next_row = row + step[0]
    next_column = column + step[1]
    # 해당 위치로 이동이 가능하다면 카운트 증가
    if 1 <= next_row <= 8 and 1 <= next_column <= 8:
        result += 1

print(result)
```

구현 예시: 문자열 재정렬

〈문제〉 문자열 재정렬: 문제 설명

- 알파벳 대문자와 숫자(0 ~ 9)로만 구성된 문자열이 입력으로 주어집니다. 이때 모든 알파벳을 오름차순으로 정렬하여 이어서 출력한 뒤에, 그 뒤에 모든 숫자를 더한 값을 이어서 출력합니다.
- 예를 들어 K1KA5CB7이라는 값이 들어오면 ABCKK13을 출력합니다.

〈문제〉 문자열 재정렬: 문제 조건

왕실의 나이트와 마찬가지로 요구사항대로 충실히 구현하면 되는 문제이다.
문자열이 입력되었을 때 문자를 하나씩 확인하고, 숫자인 경우 따로 합계를 계산하고 알파벳은 별도의 리스트에 저장한다.
결과적으로 **리스트에 저장된 알파벳은 정렬해 출력하고, 합계를 뒤에 붙여 출력하면 정답**이다.

```
data = input()
result = []
value = 0

# 문자를 하나씩 확인하며
for x in data:
    # 알파벳인 경우 결과 리스트에 삽입
    if x.isalpha():
        result.append(x)
    # 숫자는 따로 더하기
    else:
        value += int(x)

# 알파벳을 오름차순으로 정렬
result.sort()

# 숫자가 하나라도 존재하는 경우 가장 뒤에 삽입
if value != 0:
    result.append(str(value))

# 최종 결과 출력(리스트를 문자열로 변환하여 출력)
print(''.join(result))
```

영상에 나온 정답 소스 코드에 의문이 생겼다.
만약 다른 숫자 없이 숫자 0만 포함된 문자열이 들어오면 숫자가 출력되지 않는데 이건 예외 처리 안 해줘도 되나?
주석에 나온 것처럼 숫자가 하나라도 존재하는 경우 가장 뒤에 숫자를 삽입할 수 있도록 아래처럼 수정해주었다.

```python
data = input()
result = []
value = 0
data_in_num = 0

# 문자를 하나씩 확인하며
for x in data:
    # 알파벳인 경우 결과 리스트에 삽입
    if x.isalpha():
        result.append(x)
    # 숫자는 따로 더하기
    else:
        value += int(x)
        data_in_num = 1

# 알파벳을 오름차순으로 정렬
result.sort()

# 숫자가 하나라도 존재하는 경우 가장 뒤에 삽입
if data_in_num:
    result.append(str(value))

# 최종 결과 출력(리스트를 문자열로 변환하여 출력)
print(''.join(result))
```