그리디 알고리즘

- 그리디 알고리즘(탐욕법)은 **현재 상황에서 지금 당장 좋은 것만 고르는 방법**을 의미합니다.
- 일반적인 그리디 알고리즘은 문제를 풀기 위한 최소한의 아이디어를 떠올릴 수 있는 능력을 요구합니다.
- 그리디 해법은 그 정당성 분석이 중요합니다.
  - 단순히 가장 좋아 보이는 것을 반복적으로 선택해도 최적의 해를 구할 수 있는지 검토합니다.

〈문제〉 거스름 돈: 정당성 분석

- 가장 큰 화폐 단위부터 돈을 거슬러 주는 것이 최적의 해를 보장하는 이유는 무엇일까요?
  - 가지고 있는 동전 중에서 **큰 단위가 항상 작은 단위의 배수이므로 작은 단위의 동전들을 종합해 다른 해가 나올 수 없기 때문**입니다.
- 만약에 800원을 거슬러 주어야 하는데 화폐 단위가 500원, 400원, 100원이라면 어떻게 될까요?
- 그리디 알고리즘 문제에서는 이처럼 문제 풀이를 위한 최소한의 아이디어를 떠올리고 이것이 정당한지 검토할 수 있어야 합니다.

〈문제〉 거스름 돈: 시간 복잡도 분석

- 화폐의 종류가 K라고 할 때, 소스코드의 시간 복잡도는 $O(K)$입니다.
- 이 알고리즘의 시간 복잡도는 거슬러줘야 하는 금액과는 무관하며, 동전의 총 종류에만 영향을 받습니다.

〈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

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

입력 조건 • 첫째 줄에 N(1 <= N <= 100,000)과 K(2 <= K <= 100,000)가 공백을 기준으로 하여 각각 자연수로 주어집니다.

출력 조건 • 첫째 줄에 N이 1이 될 때까지 1번 혹은 2번의 과정을 수행해야 하는 횟수의 최솟값을 출력합니다.

입력 예시

```
25 5
```

출력 예시

```
2
```


〈문제〉 1이 될 때까지: 답안 예시 (Python)

```python
# N, K을 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)
```

〈문제〉 1이 될 때까지: 답안 예시 (Python)

```python
# N, K을 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)
```


〈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

```
02984
```

출력 예시 1

```
576
```

입력 예시 2

```
567
```

출력 예시 2

```
210
```


〈문제〉 곱하기 혹은 더하기: 답안 예시 (Python)

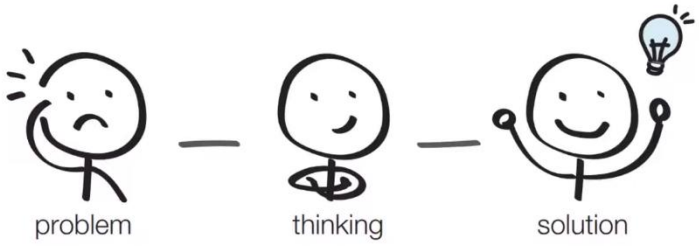
```python
data = input()

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

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

print(result)
```

〈문제〉 모험가 길드: 답안 예시 (Python)

```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(Implementation)

- 구현이란, **머릿속에 있는 알고리즘을 소스코드로 바꾸는 과정**입니다.

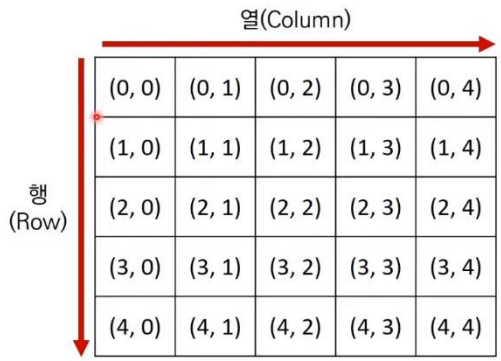


구현(Implementation)

- 흔히 알고리즘 대회에서 구현 유형의 문제란 무엇을 의미할까요?
  - **풀이를 떠올리는 것은 쉽지만 소스코드로 옮기기 어려운 문제**를 지칭합니다.
- 구현 유형의 예시는 다음과 같습니다.
  - 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

구현(Implementation)

- 일반적으로 알고리즘 문제에서의 2차원 공간은 **행렬(Matrix)**의 의미로 사용됩니다.




```
for i in range(5):
    for j in range(5):
        print('(', i, ',', j, ')', end=' ')
    print()
```

구현(Implementation)

- 시뮬레이션 및 완전 탐색 문제에서는 2차원 공간에서의 **방향 벡터**가 자주 활용됩니다.

(0, 0)	(0, 1)	(0, 2)	(0, 3)	(0, 4)
(1, 0)	(1, 1)	(1, 2)	(1, 3)	(1, 4)
(2, 0)	(2, 1)	(2, 2)	(2, 3)	(2, 4)
(3, 0)	(3, 1)	(3, 2)	(3, 3)	(3, 4)
(4, 0)	(4, 1)	(4, 2)	(4, 3)	(4, 4)

```
# 동, 북, 서, 남
dx = [0, -1, 0, 1]
dy = [1, 0, -1, 0]

# 현재 위치
x, y = 2, 2

for i in range(4):
    # 다음 위치
    nx = x + dx[i]
    ny = y + dy[i]
    print(nx, ny)
```


〈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 해결 아이디어

- 이 문제는 요구사항대로 충실히 구현하면 되는 문제입니다.
- 일련의 명령에 따라서 개체를 차례대로 이동시킨다는 점에서 **시뮬레이션(Simulation) 유형**으로도 분류되며 구현이 중요한 대표적인 문제 유형입니다.
  - 다만, 알고리즘 교재나 문제 풀이 사이트에 따라서 다르게 일컬을 수 있으므로, 코딩 테스트에서의 시뮬레이션 유형, 구현 유형, 완전 탐색 유형은 서로 유사한 점이 많다는 정도로만 기억합시다.

〈문제〉 상하좌우: 답안 예시 (Python)

```python
# 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)
```

〈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
```

〈문제〉 시각: 문제 해결 아이디어

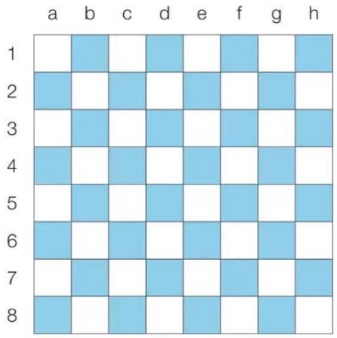
- 이 문제는 **가능한 모든 시각의 경우를 하나씩 모두 세서 풀 수 있는 문제**입니다.
- 하루는 86,400초이므로, 00시 00분 00초부터 23시 59분 59초까지의 모든 경우는 86,400가지 입니다.
  - $24 * 60 * 60 = 86{,}400$
- 따라서 단순히 시각을 1씩 증가시키면서 3이 하나라도 포함되어 있는지를 확인하면 됩니다.
- 이러한 유형은 **완전 탐색(Brute Forcing)** 문제 유형이라고 불립니다.
  - 가능한 경우의 수를 모두 검사해보는 탐색 방법을 의미합니다.

〈문제〉 시각: 답안 예시 (Python)

```python
# 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)
```

〈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: 답안 예시 (Python)

```python
# 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 next_row >= 1 and next_row <= 8 and next_column >= 1 and next_column <= 8:
        result += 1

print(result)
```

〈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: 답안 예시 (Python)

```python
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))
```