
이것이 취업을 위한 코딩 테스트다 with 파이썬

그리디 & 구현

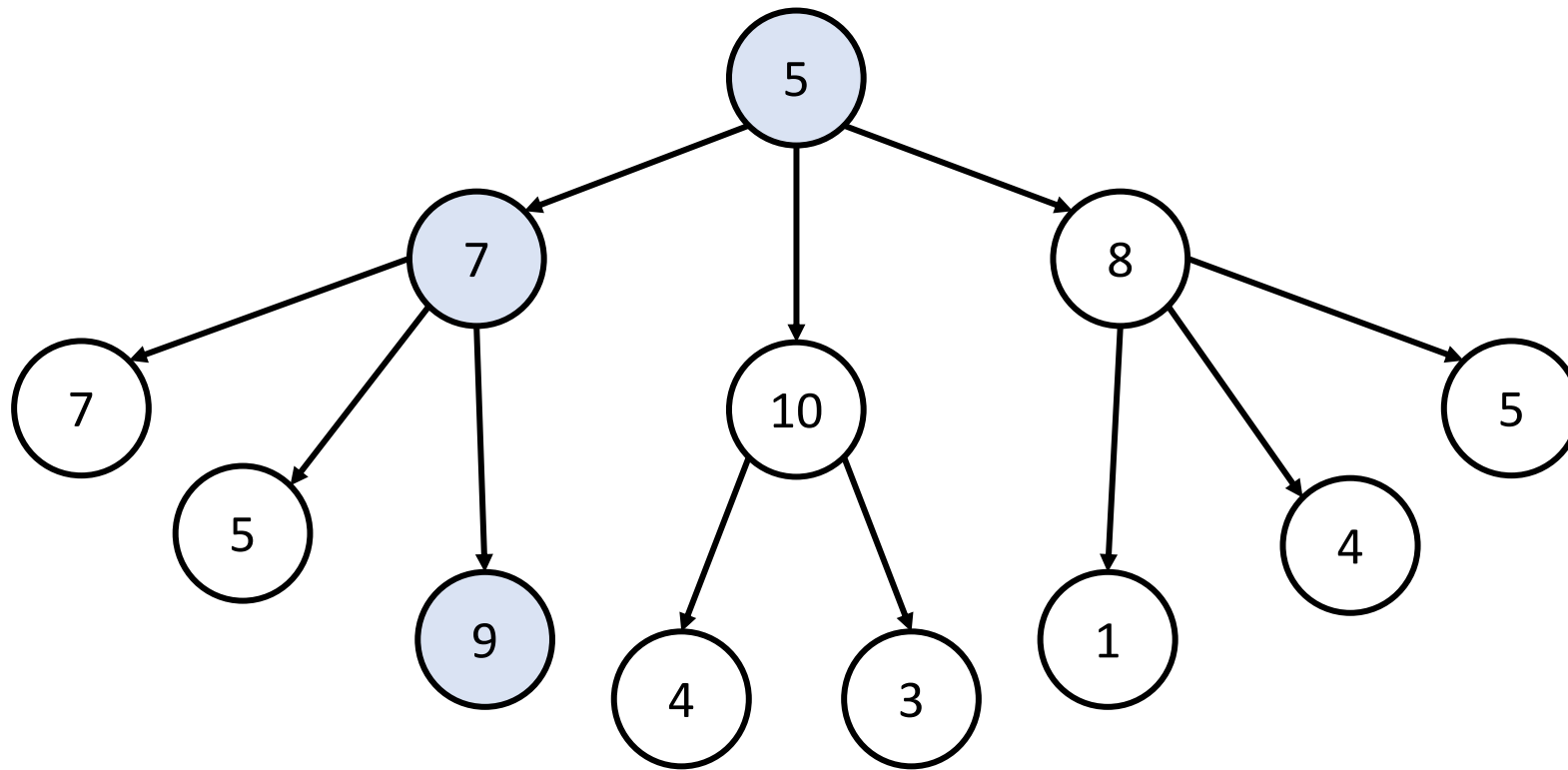
그리디 알고리즘

그리디 알고리즘

- 그리디(greedy) 알고리즘(탐욕법)은 **현재 상황에서** 지금 당장 좋은 것만 고르는 방법을 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.

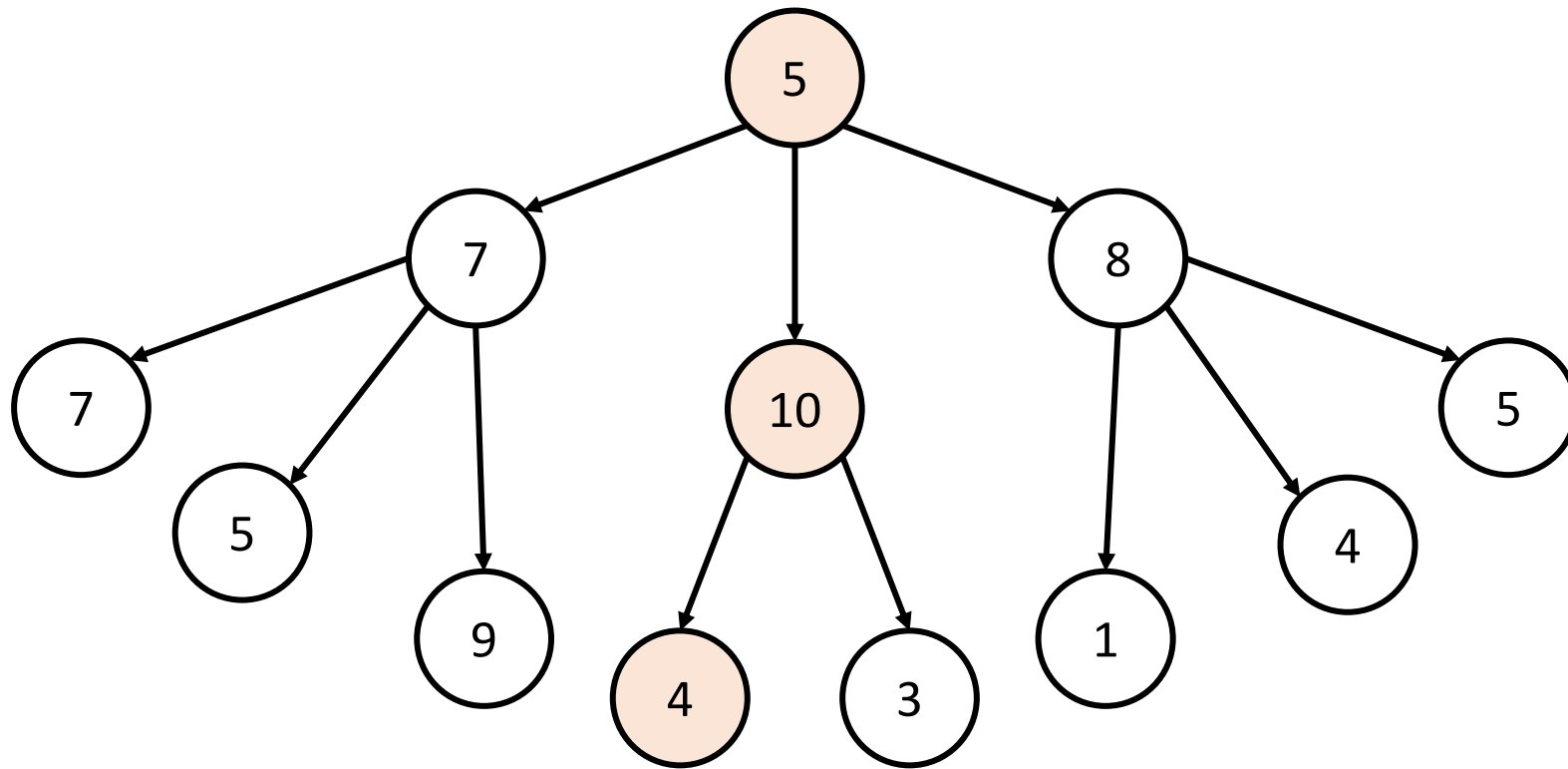
그리디 알고리즘

- [문제 상황] 루트 노드부터 시작하여 거쳐 가는 노드 값의 합을 최대로 만들고 싶습니다.
 - Q. 최적의 해는 무엇인가요?



그리디 알고리즘

- [문제 상황] 루트 노드부터 시작하여 거쳐 가는 노드 값의 합을 최대로 만들고 싶습니다.
 - Q. 단순히 매 상황에서 가장 큰 값만 고른다면 어떻게 될까요? (greedy algorithm)



그리디 알고리즘

- 일반적인 상황에서 그리디 알고리즘은 최적의 해를 보장할 수 없을 때가 많습니다.
 - 사전에 외우고 있지 않아도 풀 수 있을 가능성이 높은 문제 유형
 - 반면, 정렬, 최단경로 등의 알고리즘 유형 문제는 해당 알고리즘 사용 방법 정확히 알아야만 문제 해결 가능
- 하지만 코딩 테스트에서의 대부분의 그리디 문제는 탐욕법으로 얻은 해가 최적의 해가 되는 상황에서, 이를 추론할 수 있어야 풀리도록 출제됩니다.

<문제> 거스름 돈: 문제 설명

- 당신은 음식점의 계산을 도와주는 점원입니다. 카운터에는 거스름돈으로 사용할 500원, 100원, 50원, 10원짜리 동전이 무한히 존재한다고 가정합니다. 손님에게 거슬러 주어야 할 돈이 N 원일 때 거슬러 주어야 할 동전의 최소 개수를 구하세요. 단, 거슬러 줘야 할 돈 N 은 항상 10의 배수입니다.



<문제> 거스름 돈: 문제 해결 아이디어

- 최적의 해를 빠르게 구하기 위해서는 가장 큰 화폐 단위부터 돈을 거슬러 주면 됩니다.
- N원을 거슬러 줘야 할 때, 가장 먼저 500원으로 거슬러 줄 수 있을 만큼 거슬러 줍니다.
 - 이후에 100원, 50원, 10원짜리 동전을 차례대로 거슬러 줄 수 있을 만큼 거슬러 주면 됩니다.
- $N = 1,260$ 일 때의 예시를 확인해 봅시다.

<문제> 거스름 돈: 문제 해결 아이디어

- [Step 0] 초기 단계 – 남은 돈: 1,260원

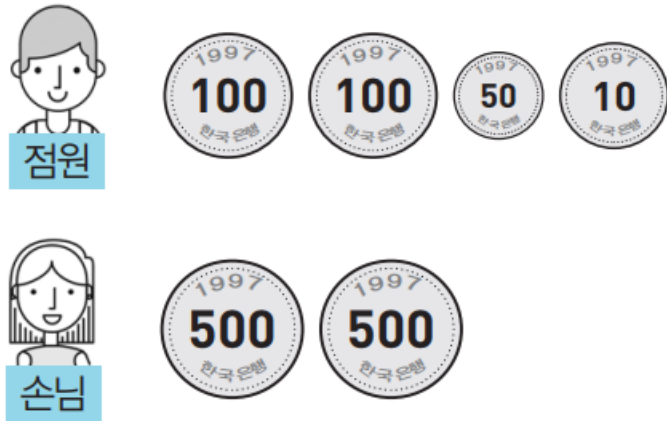


화폐 단위	500	100	50	10
손님이 받은 개수	0	0	0	0

- 본래 점원이 거슬러 줘야 할 돈은 보이지 않고 잔돈 무더기가 보여야 논리적으로 맞습니다. 이해를 돕고자 미리 거스름돈을 시각적으로 표현했습니다.

<문제> 거스름 돈: 문제 해결 아이디어

- [Step 1] 남은 돈: 260원



화폐 단위	500	100	50	10
손님이 받은 개수	2	0	0	0

<문제> 거스름 돈: 문제 해결 아이디어

- [Step 2] 남은 돈: 60원



화폐 단위	500	100	50	10
손님이 받은 개수	2	2	0	0

<문제> 거스름 돈: 문제 해결 아이디어

- [Step 3] 남은 돈: 10원



화폐 단위	500	100	50	10
손님이 받은 개수	2	2	1	0

<문제> 거스름 돈: 문제 해결 아이디어

- [Step 4] 남은 돈: 0원



(아무것도 없는 상태)



화폐 단위	500	100	50	10
손님이 받은 개수	2	2	1	1

<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 때문입니다.
- 만약에 800원을 거슬러 주어야 하는데 화폐 단위가 500원, 400원, 100원이라면 어떻게 될까요?
 - 가장 큰 단위인 500원이 작은 단위 400원의 배수가 아니라서 가장 큰 화폐 단위부터 돈을 거슬러 주는 것이 최적의 해를 보장해 주지 않는다. (다이나믹 프로그래밍 알고리즘으로 해결 가능, 교재 p226)
- 그리디 알고리즘 문제에서는 이처럼 문제 풀이를 위한 최소한의 아이디어를 떠올리고 이것이 정당한지 검토할 수 있어야 합니다.

<문제> 거스름 돈: 답안 예시 (Python)

```
n = 1260
count = 0          # 동전 갯수

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

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

print(count)
```

<문제> 거스름 돈: 답안 예시 (C++)

```
#include <bits/stdc++.h>

using namespace std;

int n = 1260;
int cnt;

int coinTypes[4] = {500, 100, 50, 10};

int main(void) {
    for (int i = 0; i < 4; i++) {
        cnt += n / coinTypes[i];
        n %= coinTypes[i];
    }
    cout << cnt << '\n';
}
```


<문제> 거스름 돈: 답안 예시 (Java)

```
public class Main {  
  
    public static void main(String[] args) {  
        int n = 1260;  
        int cnt = 0;  
        int[] coinTypes = {500, 100, 50, 10};  
  
        for (int i = 0; i < 4; i++) {  
            cnt += n / coinTypes[i];  
            n %= coinTypes[i];  
        }  
  
        System.out.println(cnt);  
    }  
}
```

<문제> 거스름 돈: 시간 복잡도 분석

- 화폐의 종류가 K 라고 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 최소 횟수를 구하라: 문제 설명

- 어떠한 수 N 이 1이 될 때까지 다음의 두 과정 중 하나를 반복적으로 선택하여 수행하려고 합니다. 단, 두 번째 연산은 N 이 K 로 나누어 떨어질 때만 선택할 수 있습니다.
 - N 에서 1을 뺍니다.
 - N 을 K 로 나눕니다.
- 예를 들어 N 이 17, K 가 4라고 가정합시다. 이때 1번의 과정을 한 번 수행하면 N 은 16이 됩니다. 이후에 2번의 과정을 두 번 수행하면 N 은 1이 됩니다. 결과적으로 이 경우 전체 과정을 실행한 횟수는 3이 됩니다. 이는 N 을 1로 만드는 최소 횟수입니다.
- N 과 K 가 주어질 때 N 이 1이 될 때까지 1번 혹은 2번의 과정을 수행해야 하는 최소 횟수를 구하는 프로그램을 작성하세요.

<문제> 1이 될 때까지: 문제 조건

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

입력 조건

- 첫째 줄에 N ($1 \leq N \leq 100,000$)과 K ($2 \leq K \leq 100,000$)가 공백을 기준으로 하여 각각 자연수로 주어집니다.

출력 조건

- 첫째 줄에 N 이 1이 될 때까지 1번 혹은 2번의 과정을 수행해야 하는 횟수의 최솟값을 출력합니다.

입력 예시

25 5

출력 예시

2

<문제> 1이 될 때까지: 문제 해결 아이디어

- 주어진 N 에 대하여 최대한 많이 나누기를 수행하면 됩니다.
- N 의 값을 줄일 때 2 이상의 수로 나누는 작업이 1을 빼는 작업보다 수를 훨씬 많이 줄일 수 있습니다.
- 예를 들어 $N = 25$, $K = 3$ 일 때는 다음과 같습니다.

단계	연산 과정	N 의 값
0단계(초기 단계)		$N = 25$
1단계	N 에서 1 빼기	$N = 24$
2단계	N 을 K 로 나누기	$N = 8$
3단계	N 에서 1 빼기	$N = 7$
4단계	N 에서 1 빼기	$N = 6$
5단계	N 을 K 로 나누기	$N = 2$
6단계	N 에서 1 빼기	$N = 1$

<문제> 1이 될 때까지: 정당성 분석

- 가능하면 최대한 많이 나누는 작업이 최적의 해를 항상 보장할 수 있을까요?
- N 이 아무리 큰 수여도, K 로 계속 나눈다면 기하급수적으로 빠르게 줄일 수 있습니다.
- 다시 말해 K 가 2 이상이기만 하면, K 로 나누는 것이 1을 빼는 것보다 항상 빠르게 N 을 줄일 수 있습니다.
 - 또한 N 은 항상 1에 도달하게 됩니다. (최적의 해 성립)

<문제> 1이 될 때까지: 답안 예시 (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 (C++)

```
#include <bits/stdc++.h>

using namespace std;

int n, k;
int result;

int main(void) {
    cin >> n >> k;
    while (true) {
        // N이 K로 나누어 떨어지는 수가 될 때까지 빼기
        int 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++;
        n /= k;
    }
    // 마지막으로 남은 수에 대하여 1씩 빼기
    result += (n - 1);
    cout << result << '\n';
}
```


<문제> 1이 될 때까지: 답안 예시 (Java)

```
import java.util.*;

public class Main {

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);

        // N, K를 공백을 기준으로 구분하여 입력 받기
        int n = sc.nextInt();
        int k = sc.nextInt();
        int result = 0;

        while (true) {
            // N이 K로 나누어 떨어지는 수가 될 때까지 빼기
            int 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);
        System.out.println(result);
    }
}
```

<문제> 곱하기 혹은 더하기: 문제 설명

- 각 자리가 숫자(0부터 9)로만 이루어진 문자열 S 가 주어졌을 때, 왼쪽부터 오른쪽으로 하나씩 모든 숫자를 확인하며 숫자 사이에 'x' 혹은 '+' 연산자를 넣어 결과적으로 만들어질 수 있는 가장 큰 수를 구하는 프로그램을 작성하세요. 단, +보다 x를 먼저 계산하는 일반적인 방식과는 달리, 모든 연산은 왼쪽에서부터 순서대로 이루어진다고 가정합니다.
- 예를 들어 "02984"라는 문자열로 만들 수 있는 가장 큰 수는 $((((0 + 2) \times 9) \times 8) \times 4) = 576$ 입니다. 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. ($1 \leq S$ 의 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: 문제 해결 아이디어

- 대부분의 경우 '+'보다는 '×'가 더 값을 크게 만듭니다.
 - 예를 들어 $5 + 6 = 11$ 이고, $5 \times 6 = 30$ 입니다.
- 다만 두 수 중에서 하나라도 '0' 혹은 '1'인 경우, 곱하기보다는 더하기를 수행하는 것이 효율적입니다.
- $0 \leq \text{어떤수} \leq 1$, 어떤수 (실수)
- 따라서 두 수에 대하여 연산을 수행할 때, 두 수 중에서 하나라도 1 이하인 경우에는 더하며, 두 수가 모두 2 이상인 경우에는 곱하면 정답입니다.

<문제> 곱하기 혹은 더하기: 답안 예시 (Python)

```
data = input()
# data = "0245"
# 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> 곱하기 혹은 더하기: 답안 예시 (C++)

```
#include <bits/stdc++.h>

using namespace std;

string str;

int main(void) {
    cin >> str;

    // 첫 번째 문자를 숫자로 변경한 값을 대입
    long long result = str[0] - '0';

    for (int i = 1; i < str.size(); i++) {
        // 두 수 중에서 하나라도 '0' 혹은 '1'인 경우, 곱하기보다는 더하기 수행
        int num = str[i] - '0';
        if (num <= 1 || result <= 1) result += num;
        else result *= num;
    }

    cout << result << '\n';
}
```

<문제> 곱하기 혹은 더하기: 답안 예시 (Java)

```
import java.util.*;

public class Main {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        String str = sc.next();

        // 첫 번째 문자를 숫자로 변경한 값을 대입
        long result = str.charAt(0) - '0';
        for (int i = 1; i < str.length(); i++) {
            // 두 수 중에서 하나라도 '0' 혹은 '1'인 경우, 곱하기보다는 더하기 수행
            int num = str.charAt(i) - '0';
            if (num <= 1 || result <= 1) {
                result += num;
            }
            else {
                result *= num;
            }
        }
        System.out.println(result);
    }
}
```

<문제> 모험가 길드: 문제 설명

- 한 마을에 모험가가 N 명 있습니다. 모험가 길드에서는 N 명의 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 X 인 모험가는 반드시 X 명 이상으로 구성된 모험가 그룹에 참여해야 여행을 떠날 수 있도록 규정했습니다.
- 동빈이는 최대 몇 개의 모험가 그룹을 만들 수 있는지 궁금합니다. N 명의 모험가에 대한 정보가 주어졌을 때, 여행을 떠날 수 있는 그룹 수의 최댓값을 구하는 프로그램을 작성하세요.

<문제> 모험가 길드: 문제 설명

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

2 3 1 2 2

- 이 경우 그룹 1에 공포도가 (1, 2, 3)인 모험가를 한 명씩 넣고, 그룹 2에 공포도가 (2, 2)인 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 N 이 주어집니다. ($1 \leq N \leq 100,000$)
- 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

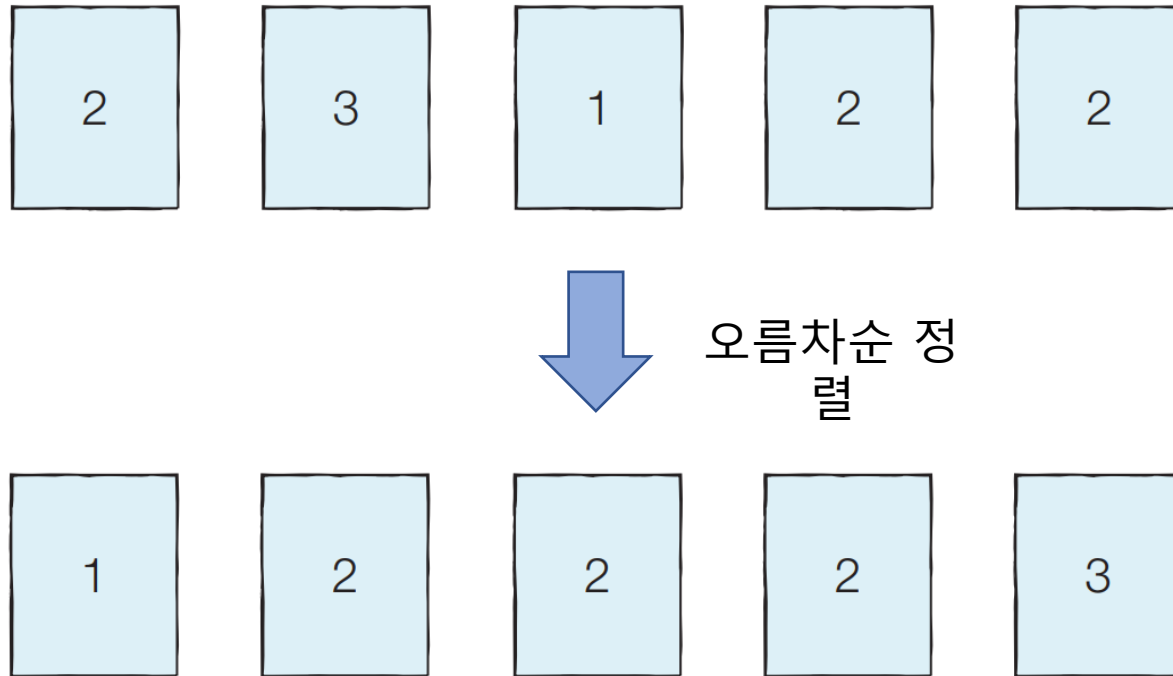
```
5
2 3 1 2 2
```

출력 예시

```
2
```

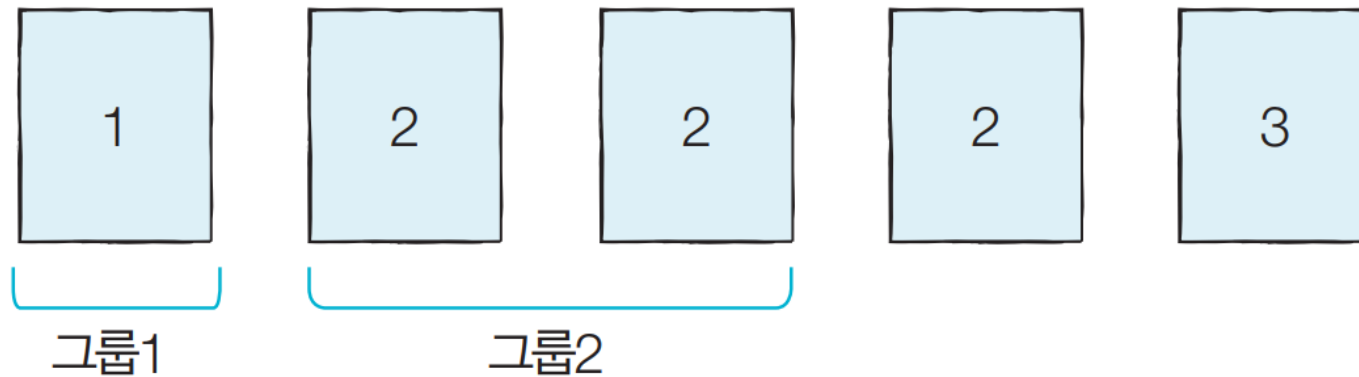
<문제> 모험가 길드: 문제 해결 아이디어

- 오름차순 정렬 이후에 공포도가 가장 낮은 모험가부터 하나씩 확인합니다.



<문제> 모험가 길드: 문제 해결 아이디어

- 앞에서부터 공포도를 하나씩 확인하며 '현재 그룹에 포함된 모험가의 수'가 '현재 확인하고 있는 공포도'보다 크거나 같다면 이를 그룹으로 설정하면 됩니다.



- 이러한 방법을 이용하면 공포도가 오름차순으로 정렬되어 있다는 점에서, 항상 최소한의 모험가의 수만 포함하여 그룹을 결성하게 됩니다.

<문제> 모험가 길드: 답안 예시 (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
```

<문제> 모험가 길드: 답안 예시 (C++)

```
#include <bits/stdc++.h>

using namespace std;

int n;
vector<int> arr;

int main(void) {
    cin >> n;
    for (int i = 0; i < n; i++) {
        int x;
        cin >> x;
        arr.push_back(x);
    }
    sort(arr.begin(), arr.end());

    int result = 0; // 총 그룹의 수
    int cnt = 0; // 현재 그룹에 포함된 모험가의 수
    for (int i = 0; i < n; i++) { // 공포도를 낮은 것부터 하나씩 확인하며
        cnt += 1; // 현재 그룹에 해당 모험가를 포함시키기
        if (cnt >= arr[i]) { // 현재 그룹에 포함된 모험가의 수가 현재의 공포도 이상이라면, 그룹 결성
            result += 1; // 총 그룹의 수 증가시키기
            cnt = 0; // 현재 그룹에 포함된 모험가의 수 초기화
        }
    }
    cout << result << '\n'; // 총 그룹의 수 출력
}
```

<문제> 모험가 길드: 답안 예시 (Java)

```
import java.util.*;

public class Main {

    public static int n;
    public static ArrayList<Integer> arrayList = new ArrayList<>();

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        n = sc.nextInt();

        for (int i = 0; i < n; i++) {
            arrayList.add(sc.nextInt());
        }
        Collections.sort(arrayList);

        int result = 0; // 총 그룹의 수
        int count = 0; // 현재 그룹에 포함된 모험가의 수
        for (int i = 0; i < n; i++) { // 공포도를 낮은 것부터 하나씩 확인하며
            count += 1; // 현재 그룹에 해당 모험가를 포함시키기
            if (count >= arrayList.get(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
            }
        }
        System.out.println(result);
    }
}
```

<기출문제-카카오> 무지의 먹방 라이브러리 (Python)

* 효율성 테스트에 부분 점수가 있는 문제입니다.

평소 식욕이 왕성한 무지는 자신의 재능을 뽐내고 싶어 쪼고 고민 끝에 카카오 TV 라이브로 방송을 하기로 마음먹었다.



그냥 먹방을 하면 다른 방송과 차별성이 없기 때문에 무지는 아래와 같이 독특한 방식을 생각해냈다.

그냥 먹방을 하면 다른 방송과 차별성이 없기 때문에 무지는 아래와 같이 독특한 방식을 생각해냈다.

회전판에 먹어야 할 N 개의 음식이 있다.

각 음식에는 1부터 N 까지 번호가 붙어있으며, 각 음식을 섭취하는데 일정 시간이 소요된다.

무지는 다음과 같은 방법으로 음식을 섭취한다.

- 무지는 1번 음식부터 먹기 시작하며, 회전판은 번호가 증가하는 순서대로 음식을 무지 앞으로 가져다 놓는다.
- 마지막 번호의 음식을 섭취한 후에는 회전판에 의해 다시 1번 음식이 무지 앞으로 온다.
- 무지는 음식 하나를 1초 동안 섭취한 후 남은 음식은 그대로 두고, 다음 음식을 섭취한다.
 - 다음 음식이란, 아직 남은 음식 중 다음으로 섭취해야 할 가장 가까운 번호의 음식을 말한다.
- 회전판이 다음 음식을 무지 앞으로 가져오는데 걸리는 시간은 없다고 가정한다.

무지가 먹방을 시작한 지 K 초 후에 네트워크 장애로 인해 방송이 잠시 중단되었다.

무지는 네트워크 정상화 후 다시 방송을 이어갈 때, 몇 번 음식부터 섭취해야 하는지를 알고자 한다.

각 음식을 모두 먹는데 필요한 시간이 담겨있는 배열 `food_times`, 네트워크 장애가 발생한 시간 K 초가 매개변수로 주어질 때 몇 번 음식부터 다시 섭취하면 되는지 `return` 하도록 `solution` 함수를 완성하라.

제한사항

- `food_times` 는 각 음식을 모두 먹는데 필요한 시간이 음식의 번호 순서대로 들어있는 배열이다.
- `k` 는 방송이 중단된 시간을 나타낸다.
- 만약 더 섭취해야 할 음식이 없다면 `-1` 을 반환하면 된다.

<기출문제-카카오> 무지의 먹방 라이브러리 (Python)

정확성 테스트 제한 사항

- food_times 의 길이는 1 이상 2,000 이하이다.
- food_times 의 원소는 1 이상 1,000 이하의 자연수이다.
- k는 1 이상 2,000,000 이하의 자연수이다.

효율성 테스트 제한 사항

- food_times 의 길이는 1 이상 200,000 이하이다.
- food_times 의 원소는 1 이상 100,000,000 이하의 자연수이다.
- k는 1 이상 2×10^{13} 이하의 자연수이다.

입출력 예

food_times	k	result
[3, 1, 2]	5	1

입출력 예 설명

입출력 예 #1

- 0~1초 동안에 1번 음식을 섭취한다. 남은 시간은 [2,1,2] 이다.
- 1~2초 동안 2번 음식을 섭취한다. 남은 시간은 [2,0,2] 이다.
- 2~3초 동안 3번 음식을 섭취한다. 남은 시간은 [2,0,1] 이다.
- 3~4초 동안 1번 음식을 섭취한다. 남은 시간은 [1,0,1] 이다.
- 4~5초 동안 (2번 음식은 다 먹었으므로) 3번 음식을 섭취한다. 남은 시간은 [1,0,0] 이다.
- 5초에서 네트워크 장애가 발생했다. 1번 음식을 섭취해야 할 때 중단되었으므로, 장애 복구 후에 1번 음식부터 다시 먹기 시작하면 된다.

Key Idea :

1. 우선순위 큐를 사용하여 (음식시간, 음식번호) tuple을 차례대로 입력
2. 우선순위 큐에서 최소 heap기준으로 1개씩 pop하면서 k보다 시간이 가장 적게 걸리는 음식들을 차례대로 제거
3. 더 이상 제거할 음식이 없다면 최종적으로 남아 있는 음식들에 대하여 k초 장애후에 먹을 음식번호 계산

<기출문제-카카오> 무지의 먹방 라이브러리 (Python)

```
def solution(food_times, k):
    # 전체 음식을 먹는 시간보다 k가 크거나 같다면 -1
    if sum(food_times) <= k:
        return -1

    # 시간이 작은 음식부터 빼야 하므로 우선순위 큐를 이용
    q = []
    for i in range(len(food_times)):
        # (음식 시간, 음식 번호) 형태로 우선순위 큐에 삽입
        heapq.heappush(q, (food_times[i], i + 1))

    sum_value = 0 # 먹기 위해 사용한 시간
    previous = 0 # 직전에 다 먹은 음식 시간
    length = len(food_times) # 남은 음식의 개수

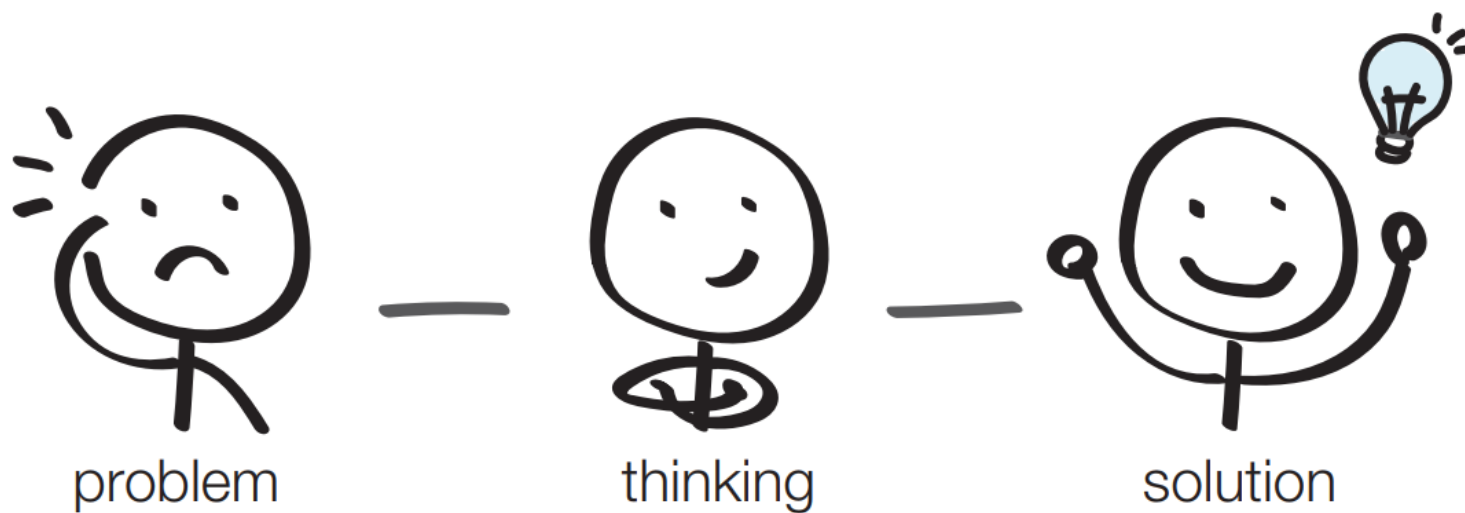
    # sum_value + (현재의 음식 시간 - 이전 음식 시간) * 현재 음식 개수와 k 비교
    while sum_value + ((q[0][0] - previous) * length) <= k:
        now = heapq.heappop(q)[0]
        sum_value += (now - previous) * length
        length -= 1 # 다 먹은 음식 제외
        previous = now # 이전 음식 시간 재설정

    # 남은 음식 중에서 몇 번째 음식인지 확인하여 출력
    result = sorted(q, key=lambda x: x[1]) # 음식의 번호 기준으로 정렬
    return result[(k - sum_value) % length][1]
```

구현: 시뮬레이션과 완전 탐색

구현(Implementation)

- 구현이란, 머릿속에 있는 알고리즘을 소스코드로 바꾸는 과정입니다.
- Solution찾기 : 작은 프로그램 (1~1주일), 크고 복잡한 프로그램 (1달까지 걸릴 수 있고, 여러명의 팀원이 서로 brainstorming 등을 통해 의견 교환해서 최적의 알고리즘을 만들어 내야 됨)

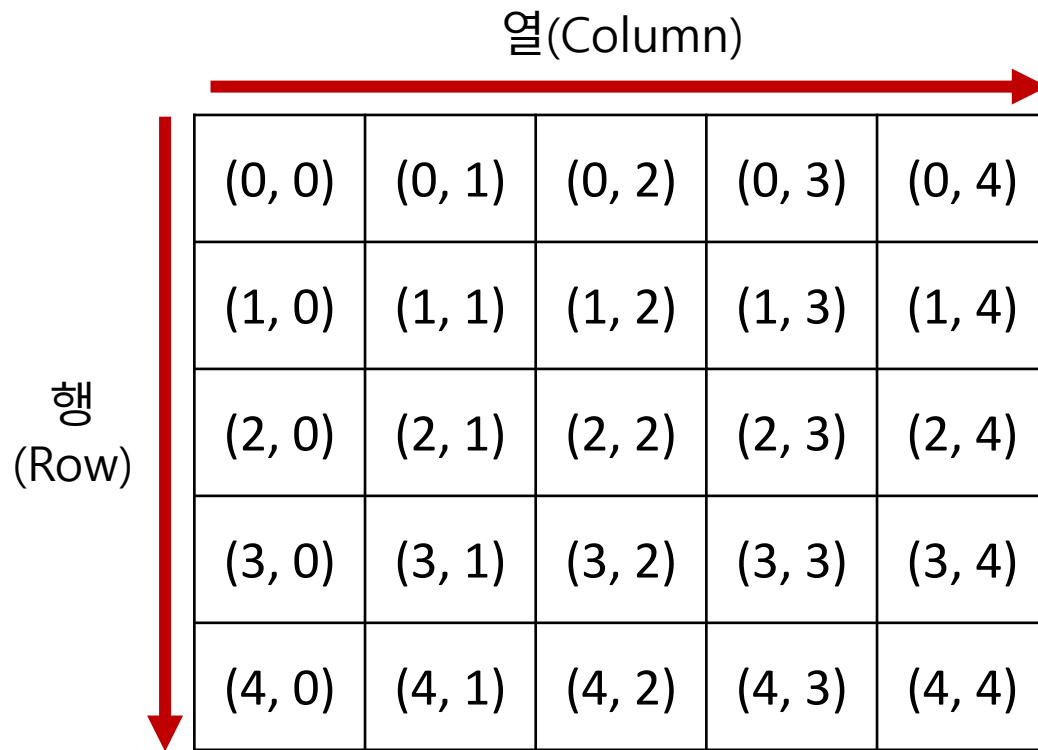


구현(Implementation)

- 흔히 알고리즘 대회에서 구현 유형의 문제란 무엇을 의미할까요?
 - 풀이를 떠올리는 것은 쉽지만 소스코드로 옮기기 어려운 문제를 지칭합니다.
- 시뮬레이션(Simulation)과 완전탐색(Exhaustive Search) 유형도 구현 유형에 포함됨
 - 시뮬레이션 : 문제에서 제시한 알고리즘을 한 단계씩 차례대로 직접 수행
 - 완전탐색 : 모든 경우의 수를 빠짐없이 다 계산하여 해결하는 방법
- 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.



The diagram shows a 5x5 matrix with rows and columns indexed from 0 to 4. A red arrow labeled '열(Column)' points to the right above the matrix. A red arrow labeled '행 (Row)' points downwards to the left of the matrix.

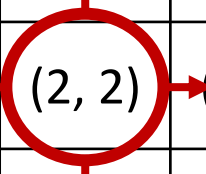
(0, 0)	(0, 1)	(0, 2)	(0, 3)	(0, 4)
(1, 0)	(1, 1)	(1, 2)	(1, 3)	(1, 4)
(2, 0)	(2, 1)	(2, 2)	(2, 3)	(2, 4)
(3, 0)	(3, 1)	(3, 2)	(3, 3)	(3, 4)
(4, 0)	(4, 1)	(4, 2)	(4, 3)	(4, 4)

```
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 벡터가 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
```

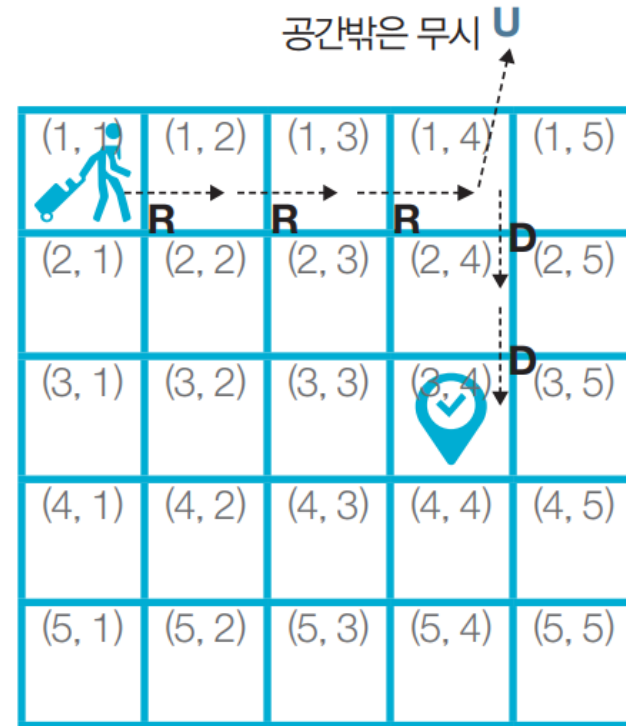
```
for i in range(4):  
    # 다음 위치  
    nx = x + dx[i]  
    ny = y + dy[i]  
    print(nx, ny)
```

<문제> 상하좌우: 문제 설명

- 여행가 A는 $N \times N$ 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 이동할 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 $N \times N$ 크기의 정사각형 공간을 벗어나는 움직임은 무시됩니다. 예를 들어 (1, 1)의 위치에서 L 혹은 U를 만나면 무시됩니다. 다음은 $N = 5$ 인 지도와 계획서입니다.



<문제> 상하좌우: 문제 조건

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

입력 조건

- 첫째 줄에 공간의 크기를 나타내는 N 이 주어집니다. ($1 \leq N \leq 100$)
- 둘째 줄에 여행가 A가 이동할 계획서 내용이 주어집니다. ($1 \leq$ 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
```

<문제> 상하좌우: 문제 해결 아이디어

- 이 문제는 요구사항대로 충실히 구현하면 되는 문제입니다.
- 일련의 명령에 따라서 개체를 차례대로 이동시킨다는 점에서 시뮬레이션(Simulation) 유형으로도 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)

```
# 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> 상하좌우: 답안 예시 (C++)

```
#include <bits/stdc++.h>

using namespace std;

int n;
string plans;
int x = 1, y = 1;

// L, R, U, D에 따른 이동 방향
int dx[4] = {0, 0, -1, 1};
int dy[4] = {-1, 1, 0, 0};
char moveTypes[4] = {'L', 'R', 'U', 'D'};
```

```
int main(void) {
    cin >> n;
    cin.ignore(); // 버퍼 비우기
    getline(cin, plans);
    // 이동 계획을 하나씩 확인하기
    for (int i = 0; i < plans.size(); i++) {
        char plan = plans[i];
        // 이동 후 좌표 구하기
        int nx = -1, ny = -1;
        for (int j = 0; j < 4; j++) {
            if (plan == moveTypes[j]) {
                nx = x + dx[j];
                ny = y + dy[j];
            }
        }
        // 공간을 벗어나는 경우 무시
        if (nx < 1 || ny < 1 || nx > n || ny > n) continue;
        // 이동 수행
        x = nx;
        y = ny;
    }
    cout << x << ' ' << y << '\n';
    return 0;
}
```

<문제> 시각: 문제 설명

- 정수 N 이 입력되면 00시 00분 00초부터 N 시 59분 59초까지의 모든 시각 중에서 3이 하나라도 포함되는 모든 경우의 수를 구하는 프로그램을 작성하세요. 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 N 이 입력됩니다. ($0 \leq N \leq 23$)

출력 조건

- 00시 00분 00초부터 N 시 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 문제입니다.
- 하루는 86,400초이므로, 00시 00분 00초부터 23시 59분 59초까지의 모든 경우는 86,400가지입니다.
 - $24 * 60 * 60 = 86,400$
- 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)

```
# H 입력 받기
h = int(input())

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

print(count)
```

<문제> 시각: 답안 예시 (C++)

```
#include <bits/stdc++.h>

using namespace std;

int h, cnt;

// 특정한 시각 안에 '3'이 포함되어 있는지의 여부
bool check(int h, int m, int s) {
    if (h % 10 == 3 || m / 10 == 3 || m % 10 == 3 || s / 10 == 3 || s % 10 == 3)
        return true;
    return false;
}

int main(void) {
    // H 입력 받기
    cin >> h;
    for (int i = 0; i <= h; i++) {
        for (int j = 0; j < 60; j++) {
            for (int k = 0; k < 60; k++) {
                if (check(i, j, k)) cnt++;
            }
        }
    }
    cout << cnt << 'Wn';
    return 0;
}
```

<문제> 시각: 답안 예시 (Java)

```
import java.util.*;

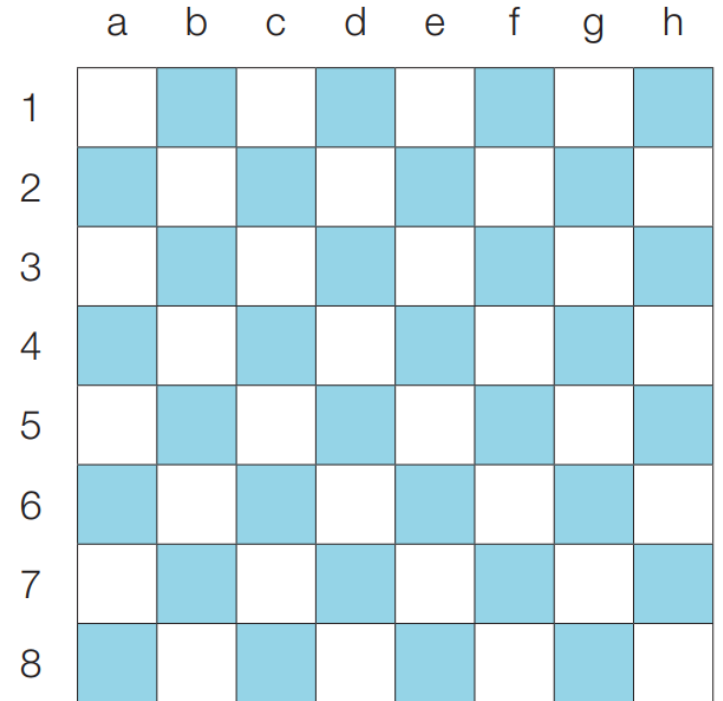
public class Main {
    // 특정한 시각 안에 '3'이 포함되어 있는지의 여부
    public static boolean check(int h, int m, int s) {
        if (h % 10 == 3 || m / 10 == 3 || m % 10 == 3 || s / 10 == 3 || s % 10 == 3)
            return true;
        return false;
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);

        // H를 입력받기
        int h = sc.nextInt();
        int cnt = 0;
        for (int i = 0; i <= h; i++) {
            for (int j = 0; j < 60; j++) {
                for (int k = 0; k < 60; k++) {
                    // 매 시각 안에 '3'이 포함되어 있다면 카운트 증가
                    if (check(i, j, k)) cnt++;
                }
            }
        }
        System.out.println(cnt);
    }
}
```

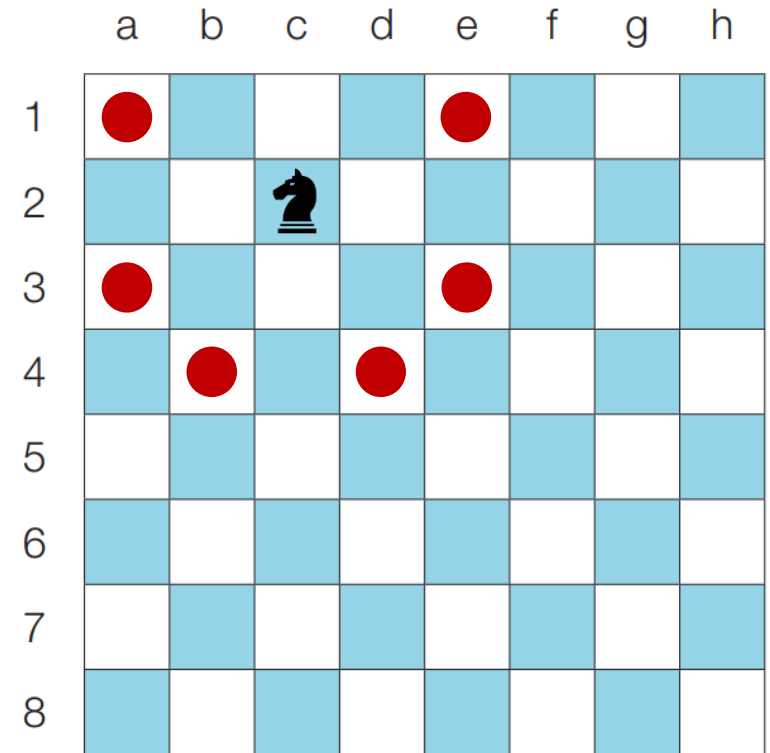
<문제> 왕실의 나이트: 문제 설명

- 행복 왕국의 왕실 정원은 체스판과 같은 8×8 좌표 평면입니다. 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 밖으로 는 나갈 수 없습니다.
- 나이트는 특정 위치에서 다음과 같은 2가지 경우로 이동할 수 있
1. 수평으로 두 칸 이동한 뒤에 수직으로 한 칸 이동하기
2. 수직으로 두 칸 이동한 뒤에 수평으로 한 칸 이동하기



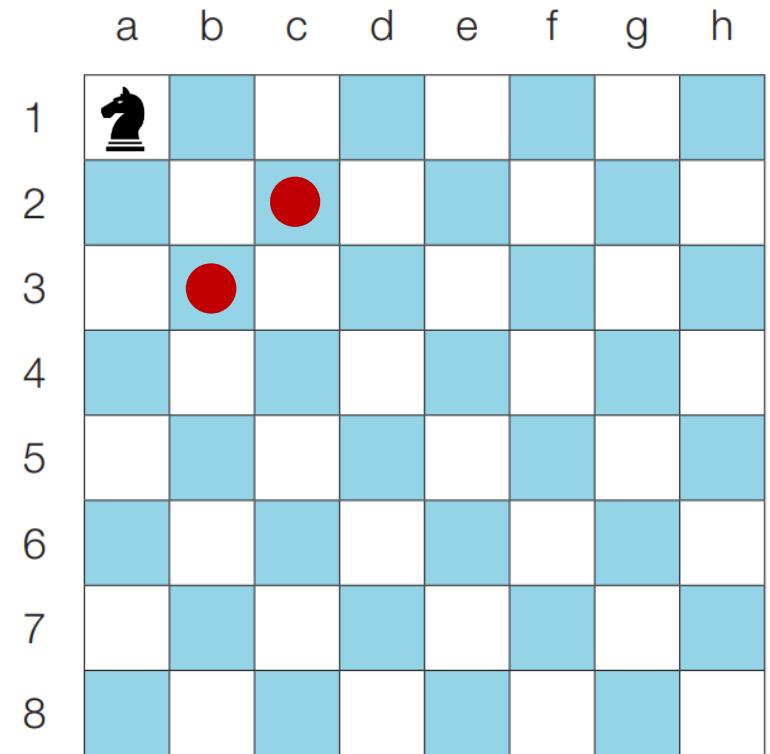
<문제> 왕실의 나이트: 문제 설명

- 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 6가지입니다



<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.
 - a1에 있을 때 이동할 수 있는 경우의 수는 2가지입니다



<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

<문제> 왕실의 나이트: 문제 해결 아이디어

- 요구사항대로 충실히 구현하면 되는 문제입니다.
- 나이트의 8가지 경로를 하나씩 확인하며 각 위치로 이동이 가능한지 확인합니다.
 - 리스트를 이용하여 8가지 방향에 대한 방향 벡터를 정의합니다.

<문제> 왕실의 나이트: 답안 예시 (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> 왕실의 나이트: 답안 예시 (C++)

```
#include <bits/stdc++.h>

using namespace std;

string inputData;
// 나이트가 이동할 수 있는 8가지 방향 정의
int dx[] = {-2, -1, 1, 2, 2, 1, -1, -2};
int dy[] = {-1, -2, -2, -1, 1, 2, 2, 1};

int main(void) {
    // 현재 나이트의 위치 입력받기
    cin >> inputData;
    int row = inputData[1] - '0';
    int column = inputData[0] - 'a' + 1;

    // 8가지 방향에 대하여 각 위치로 이동이 가능한지 확인
    int result = 0;
    for (int i = 0; i < 8; i++) {
        // 이동하고자 하는 위치 확인
        int nextRow = row + dx[i];
        int nextColumn = column + dy[i];
        // 해당 위치로 이동이 가능하다면 카운트 증가
        if (nextRow >= 1 && nextRow <= 8 && nextColumn >= 1 && nextColumn <= 8) {
            result += 1;
        }
    }
    cout << result << 'Wn';
    return 0;
}
```

<문제> 왕실의 나이트: 답안 예시 (Java)

```
import java.util.*;

public class Main {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        // 현재 나이트의 위치 입력받기
        String inputData = sc.nextLine();
        int row = inputData.charAt(1) - '0';
        int column = inputData.charAt(0) - 'a' + 1;

        // 나이트가 이동할 수 있는 8가지 방향 정의
        int[] dx = {-2, -1, 1, 2, 2, 1, -1, -2};
        int[] dy = {-1, -2, -2, -1, 1, 2, 2, 1};
        // 8가지 방향에 대하여 각 위치로 이동이 가능한지 확인
        int result = 0;
        for (int i = 0; i < 8; i++) {
            // 이동하고자 하는 위치 확인
            int nextRow = row + dx[i];
            int nextColumn = column + dy[i];
            // 해당 위치로 이동이 가능하다면 카운트 증가
            if (nextRow >= 1 && nextRow <= 8 && nextColumn >= 1 && nextColumn <= 8) {
                result += 1;
            }
        }
        System.out.println(result);
    }
}
```

<문제> 게임 개발 : 문제 설명

- 현민이는 게임 캐릭터가 맵 안에서 움직이는 시스템을 개발 중이다. 캐릭터가 있는 장소는 1×1 크기의 정사각형으로 이뤄진 $N \times M$ 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.
 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 회전한 다음 왼쪽으로 한 칸을 전진한다. 왼쪽 방향에 가보지 않은 칸이 없다면, 왼쪽 방향으로 회전만 수행하고 1단계로 돌아간다.

<문제> 게임 개발 : 문제 설명 (계속)

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

<문제> 게임개발: 문제 조건

난이도 ●●○ | 풀이 시간 40분 | 시간 제한 1초 | 메모리 제한 128MB

입력 조건

첫째 줄에 맵의 세로 크기 N 과 가로 크기 M 을 공백으로 구분하여 입력한다.

($3 \leq N, M \leq 50$)

둘째 줄에 게임 캐릭터가 있는 칸의 좌표 (A, B)와 바라보는 방향 d 가 각각 서로 공백으로 구분하여 주어진다. 방향 d 의 값으로는 다음과 같이 4가지가 존재한다.

0 : 북쪽

1 : 동쪽

2 : 남쪽

3 : 서쪽

셋째 줄부터 맵이 육지인지 바다인지에 대한 정보가 주어진다. N 개의 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 조건

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

<문제> 게임 개발 : 문제 해결 아이디어

- 전형적인 시뮬레이션 문제입니다.
 - 삼성전자 공채 코딩테스트에서 자주 출제되는 대표적인 유형
- 별도의 알고리즘이 필요하기보다는 문제에서 요구하는 내용을 오류 없이 성실하게 구현
- 결과적으로 리스트에 저장된 알파벳을 오름차순 정렬(sorting)해 출력하고, 합계를 뒤에 붙여 출력하면 정답입니다.

<문제> 게임개발: 답안 예시 (Python)

```
# N, M을 공백을 기준으로 구분하여 입력받기
n, m = map(int, input().split())

# 방문한 위치를 저장하기 위한 맵을 생성하여 0으로 초기화
d = [[0] * m for _ in range(n)]
# 현재 캐릭터의 X 좌표, Y 좌표, 방향을 입력받기
x, y, direction = map(int, input().split())
d[x][y] = 1 # 현재 좌표 방문 처리

# 전체 맵 정보를 입력받기
array = []
for i in range(n):
    array.append(list(map(int, input().split())))

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

# 왼쪽으로 회전
def turn_left():
    global direction
    direction -= 1
    if direction == -1:
        direction = 3
```

```
# 시뮬레이션 시작
count = 1
turn_time = 0
while True:
    # 왼쪽으로 회전
    turn_left()
    nx = x + dx[direction]
    ny = y + dy[direction]
    # 회전한 이후 정면에 가보지 않은 칸이 존재하는 경우 이동
    if d[nx][ny] == 0 and array[nx][ny] == 0:
        d[nx][ny] = 1
        x = nx
        y = ny
        count += 1
        turn_time = 0
        continue
    # 회전한 이후 정면에 가보지 않은 칸이 없거나 바다인 경우
    else:
        turn_time += 1
    # 네 방향 모두 갈 수 없는 경우
    if turn_time == 4:
        nx = x - dx[direction]
        ny = y - dy[direction]
        # 뒤로 갈 수 있다면 이동하기
        if array[nx][ny] == 0:
            x = nx
            y = ny
        # 뒤가 바다로 막혀있는 경우
        else:
            break
        turn_time = 0
# 정답 출력
print(count)
```


<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

난이도 ●○○ | 풀이 시간 20분 | 시간 제한 1초 | 메모리 제한 128MB | 기출 Facebook 인터뷰

입력 조건 • 첫째 줄에 하나의 문자열 S가 주어집니다. ($1 \leq S$ 의 길이 $\leq 10,000$)

출력 조건 • 첫째 줄에 문제에서 요구하는 정답을 출력합니다.

입력 예시 1

K1KA5CB7

출력 예시 1

ABCKK13

입력 예시 2

AJKDLSI412K4JSJ9D

출력 예시 2

ADDIJJJKKLSS20

<문제> 문자열 재정렬: 문제 해결 아이디어

- 요구사항대로 충실히 구현하면 되는 문제입니다.
- 문자열이 입력되었을 때 문자를 하나씩 확인합니다.
 - 숫자인 경우 따로 합계를 계산합니다.
 - 알파벳은 경우 별도의 리스트에 저장합니다.
- 결과적으로 리스트에 저장된 알파벳을 오름차순 정렬(sorting)해 출력하고, 합계를 뒤에 붙여 출력하면 정답입니다.

<문제> 문자열 재정렬: 답안 예시 (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))
```

<문제> 문자열 재정렬: 답안 예시 (C++)

```
#include <bits/stdc++.h>

using namespace std;

string str;
vector<char> result;
int value = 0;
```

```
int main(void) {
    cin >> str;
    // 문자를 하나씩 확인하며
    for (int i = 0; i < str.size(); i++) {
        // 알파벳인 경우 결과 리스트에 삽입
        if (isalpha(str[i])) {
            result.push_back(str[i]);
        }
        // 숫자는 따로 더하기
        else {
            value += str[i] - '0';
        }
    }
    // 알파벳을 오름차순으로 정렬
    sort(result.begin(), result.end());
    // 알파벳을 차례대로 출력
    for (int i = 0; i < result.size(); i++) {
        cout << result[i];
    }
    // 숫자가 하나라도 존재하는 경우 가장 뒤에 출력
    if (value != 0) cout << value;
    cout << '\n';
}
```