

Alogorithm(알고리즘) 실전 문제 : 난이도 중하 연습

: 문제를 자바, 파이썬으로 풀기

문제1) 단어 글자 뒤집기

문장이 주어졌을 때, 단어를 모두 뒤집어서 출력하는 프로그램을 작성하시오. 단, 단어의 순서는 바꿀 수 없다. 단어는 영어 알파벳으로만 이루어져 있다.

입력 : 첫째 줄에 테스트 케이스의 개수 T가 주어진다. 각 테스트 케이스는 한 줄로 이루어져 있으며, 문장이 하나 주어진다. 단어의 길이는 최대 20, 문장의 길이는 최대 1000이다. 단어와 단어 사이에는 공백이 하나 있다.

출력 : 각 테스트 케이스에 대해서, 입력으로 주어진 문장의 단어를 모두 뒤집어 출력한다.

입력 : I am happy today

출력 : I ma yppah yadot

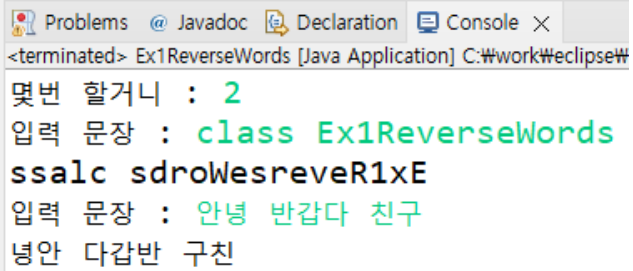
이 문제를 해결하기 위해, 각 테스트 케이스마다 문장을 단어 단위로 나누고, 각 단어를 뒤집은 후 다시 조합하는 방식을 사용할 수 있습니다. 먼저 자바로 구현해 보겠습니다.

```
import java.util.Scanner;

public class Ex1ReverseWords {
    public static void main(String[] args) {
        System.out.print("몇번 할거니 : ");
        Scanner scanner = new Scanner(System.in);
        int T = Integer.parseInt(scanner.nextLine()); // 테스트 케이스 개수 읽기

        for(int i = 0; i < T; i++) {
            System.out.print("입력 문장 : ");
            String sentence = scanner.nextLine(); // 문장 읽기
            String[] words = sentence.split(" "); // 공백으로 단어 분리

            for(String word : words) {
                System.out.print(new StringBuilder(word).reverse().toString() + " ");
            }
            System.out.println(); // 단어 출력 후 개행
        }
        scanner.close();
    }
}
```



```

<terminated> Ex1ReverseWords [Java Application] C:\work\weclipse\#
몇번 할거니 : 2
입력 문장 : class Ex1ReverseWords
ssalc sdrowesreveR1xE
입력 문장 : 안녕 반갑다 친구
녕안 다갑반 구친

```

파이썬으로 변경한 코드는 다음과 같습니다. 자바 코드의 구조를 따라 입력 횟수와 각 문장을 받아서 처리하는 방식으로 구현했습니다.

python

Copy code

```
print("몇번 할거니 : ", end="")
```

```
T = int(input()) # 테스트 케이스 개수
```

```
for _ in range(T):
```

```
    print("입력 문장 : ", end="")
```

```
    sentence = input() # 문장 입력
```

```
    words = sentence.split(" ") # 공백으로 단어 분리
```

```
    reversed_words = ' '.join([word[::-1] for word in words]) # 단어를 뒤집고 공백으로 재결합
```

```
    print(reversed_words) # 뒤집은 단어 출력
```

자바에서 사용한 Scanner 클래스와 비슷한 입력 방식을 파이썬에서는 input() 함수를 사용하여 구현했습니다. 또한, 파이썬의 문자열 슬라이싱 기능을 사용하여 각 단어를 뒤집었습니다. 이후 뒤집은 단어들을 ' '.join() 메서드를 사용하여 공백으로 구분해 하나의 문자열로 재결합하고, 이를 출력합니다.

문제2) 요세푸스 문제

1번부터 N번까지 N명의 사람이 원을 이루면서 앉아있고, 양의 정수 $K(\leq N)$ 가 주어진다.

이제 순서대로 K번째 사람을 제거한다. 한 사람이 제거되면 남은 사람들로 이루어진 원을 따라 이 과정을 계속해 나간다.

이 과정은 N명의 사람이 모두 제거될 때까지 계속된다. 원에서 사람들이 제거되는 순서를 (N, K)-요세푸스 순열이라고 한다.

예를 들어 (7, 3)-요세푸스 순열은 $\langle 3, 6, 2, 7, 5, 1, 4 \rangle$ 이다.

N과 K가 주어지면 (N, K)-요세푸스 순열을 구하는 프로그램을 작성하시오.

입력 : 첫째 줄에 N과 K가 빈 칸을 사이에 두고 순서대로 주어진다. ($1 \leq K \leq N \leq 5,000$)

출력 : 예제와 같이 요세푸스 순열을 출력한다.

예제 입력 1 : 7 3

예제 출력 1 : $\langle 3, 6, 2, 7, 5, 1, 4 \rangle$

자바로 푸는 방법을 설명해 드리겠습니다. 요세푸스 문제는 큐(Queue) 자료구조를 사용하여 효율적으로 해결할 수 있습니다. 자바에서는 LinkedList 클래스를 사용하여 큐를 구현할 수 있습니다. LinkedList는 Queue 인터페이스를 구현하고 있기 때문에, 큐로서의 기능을 제공합니다. 이를 이용하여 요세푸스 순열을 구하는 과정은 다음과 같습니다:

1부터 N까지의 숫자를 순서대로 큐에 삽입합니다.

큐에서 숫자를 하나씩 꺼내서 바로 그 숫자를 다시 큐의 끝에 삽입합니다. 이 과정을 K-1번 반복합니다.

K번째로 꺼낸 숫자는 큐에 다시 삽입하지 않고 결과 순열에 추가합니다.

큐가 비어있을 때까지 2번과 3번 과정을 반복합니다.

자바 코드로 구현하면 다음과 같습니다.

```
package pack;

import java.util.LinkedList; // LinkedList 라이브러리 import
import java.util.Queue; // Queue 인터페이스 import
import java.util.Scanner; // 사용자 입력을 받기 위한 Scanner 라이브러리 import

public class Ex2JosephusProblem {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in); // 사용자 입력을 받기 위한 Scanner 객체 생성
        System.out.print("원에 앉아있는 사람의 수와 제거될 순서를 입력 (7 3):");
        int N = scanner.nextInt(); // N을 입력 받음, 즉 원에 앉아있는 사람의 수
        int K = scanner.nextInt(); // K를 입력 받음, 제거될 순서

        Queue<Integer> queue = new LinkedList<>(); // Integer 타입의 LinkedList로 Queue 구현
        for (int i = 1; i <= N; i++) {
            queue.add(i); // 1부터 N까지의 숫자를 큐에 차례대로 삽입
        }
    }
}
```

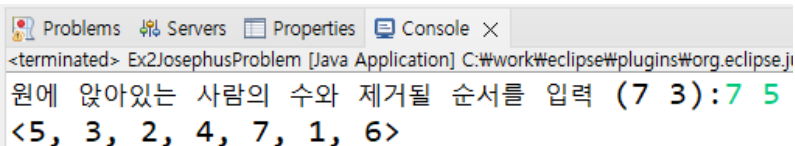
```

StringBuilder sb = new StringBuilder(); // 결과를 저장할 StringBuilder 객체 생성
sb.append('<'); // 요세푸스 순열 출력 형식 시작

while (!queue.isEmpty()) { // 큐가 빌 때까지 반복
    for (int i = 0; i < K - 1; i++) {
        queue.add(queue.poll()); // K-1번째까지는 큐에서 꺼낸 후 다시 큐에 삽입
    }
    sb.append(queue.poll()); // K번째는 큐에서 꺼내고 결과 문자열에 추가
    if (!queue.isEmpty()) { // 큐가 비어있지 않다면 숫자 사이에 쉼표와 공백 추가
        sb.append(", ");
    }
}

sb.append('>'); // 요세푸스 순열 출력 형식 종료
System.out.println(sb.toString()); // 최종 결과 출력
}
}

```



```

<terminated> Ex2JosephusProblem [Java Application] C:\work\weclipse\plugins\org.eclipse.j
원에 앉아있는 사람의 수와 제거될 순서를 입력 (7 3):7 5
<5, 3, 2, 4, 7, 1, 6>

```

자바로 구현한 요세푸스 문제를 파이썬으로 변환하면 다음과 같습니다. 파이썬에서는 리스트(list)를 사용하여 큐(Queue)의 기능을 비슷하게 구현할 수 있습니다.

```
from collections import deque
```

```

def josephus_problem(N, K):
    # 초기 설정: 1부터 N까지의 숫자를 담은 큐 생성
    queue = deque(range(1, N + 1))
    result = []

    while queue:
        # K-1번째까지는 큐의 맨 앞의 요소를 큐의 맨 뒤로 보냄
        for _ in range(K - 1):
            queue.append(queue.popleft())
        # K번째 요소는 큐에서 제거하고 결과 리스트에 추가
        result.append(queue.popleft())

    # 결과 리스트를 문자열 형식으로 변환하여 반환
    return "<" + ", ".join(map(str, result)) + ">"

```

```
# 예제 입력
```

```
N, K = map(int, input().split())
```

```
# 요세푸스 순열 출력
```

```
print(josephus_problem(N, K))
```

이 코드는 다음과 같이 작동합니다:

`collections.deque`를 사용하여 큐를 초기화합니다. `deque`는 양방향 큐로, 리스트에 비해 앞뒤 양쪽에서의 추가와 제거가 더 효율적입니다.

입력 받은 `N`과 `K`에 대하여, `N`개의 숫자(1부터 `N`까지)를 `deque`에 넣습니다.

`K-1`번째 사람까지는 `deque`의 맨 앞에서 꺼내 맨 뒤로 옮깁니다(`popleft`와 `append` 메소드 사용).

`K`번째 사람은 `deque`에서 완전히 제거하고(`popleft`), 결과 리스트에 추가합니다.

이 과정을 `deque`가 비어있을 때까지 반복합니다.

최종적으로 구한 요세푸스 순열을 문자열로 변환하여 출력합니다.

문제3) 문자열 분석

문자열 N개가 주어진다. 이때, 문자열에 포함되어 있는 소문자, 대문자, 숫자, 공백의 개수를 구하는 프로그램을 작성하시오.

각 문자열은 알파벳 소문자, 대문자, 숫자, 공백으로만 이루어져 있다.

입력 : 첫째 줄부터 N번째 줄까지 문자열이 주어진다. ($1 \leq N \leq 100$) 문자열의 길이는 100을 넘지 않는다.

출력 : 첫째 줄부터 N번째 줄까지 각각의 문자열에 대해서 소문자, 대문자, 숫자, 공백의 개수를 공백으로 구분해 출력한다.

예제 입력 1 : This is String

예제 출력 1 : 10 2 0 2

예제 입력 1 : 0L1A2S3T4L5I6N7E8

예제 출력 1 : 0 8 9 0

이 문제를 자바로 해결하기 위해서는 입력 받은 각 문자열을 분석하여 소문자, 대문자, 숫자, 공백의 개수를 찾아야 합니다. 자바에서 문자열과 관련된 처리는 매우 흔하므로, 문자의 종류를 판별하기 위한 조건문과 문자열을 순회할 수 있는 반복문을 사용할 것입니다.

접근 방법 :

Scanner 클래스를 사용해 사용자로부터 문자열 입력을 받습니다.

입력받은 문자열에 대해 문자 하나하나를 순회하며, 각 문자가 소문자, 대문자, 숫자, 공백 중 어디에 해당하는지 확인합니다.

해당하는 카테고리의 카운트를 증가시킵니다.

모든 문자의 검사가 끝난 후, 각 카테고리별 카운트를 출력합니다.

주의 사항 :

입력의 마지막을 알기 위해서는 입력이 더 이상 없을 때까지 반복 읽어야 합니다. 이를 위해 Scanner 클래스의 hasNextLine() 메소드를 사용할 수 있습니다.

자바에서 문자의 종류를 판별하기 위해 Character 클래스의 isLowerCase(), isUpperCase(), isDigit(), isWhitespace() 등의 메소드를 활용할 수 있습니다.

구현 코드 :

```
package pack;
import java.util.Scanner;

public class Ex3StringAnal {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("데이터 입력 : ");

        while (scanner.hasNextLine()) { // 입력이 있을 때까지 계속 읽음
            String line = scanner.nextLine(); // 한 줄을 읽음
            int lowercase = 0, uppercase = 0, digits = 0, whitespaces = 0;
```

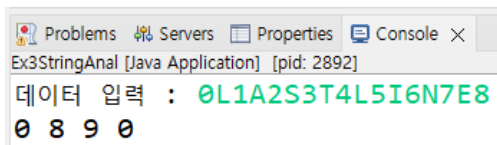
```

for (char ch : line.toCharArray()) { // 문자열을 순회하며 각 문자 확인
    if (Character.isLowerCase(ch)) { // 소문자인 경우
        lowercase++;
    } else if (Character.isUpperCase(ch)) { // 대문자인 경우
        uppercase++;
    } else if (Character.isDigit(ch)) { // 숫자인 경우
        digits++;
    } else if (Character.isWhitespace(ch)) { // 공백인 경우
        whitespaces++;
    }
}

// 소문자, 대문자, 숫자, 공백의 개수를 출력
System.out.println(lowercase + " " + uppercase + " " + digits + " " + whitespaces);
}

scanner.close(); // 스캐너 사용 종료
}
}

```



이 코드는 문자열을 입력 받고, 각 문자열에 대해 소문자, 대문자, 숫자, 공백의 개수를 계산한 다음 출력합니다. 사용자가 더 이상 입력하지 않고 입력의 끝을 알릴 때까지 (예를 들어, 입력 스트림의 끝에 도달할 때까지) 이 과정을 반복합니다.

다음은 주어진 문제를 해결하기 위한 파이썬 코드입니다:

python

Copy code

while True:

try:

```

line = input() # 사용자로부터 한 줄 입력 받음
lowercase = sum(c.islower() for c in line) # 소문자 개수
uppercase = sum(c.isupper() for c in line) # 대문자 개수
digits = sum(c.isdigit() for c in line) # 숫자 개수
whitespaces = sum(c.isspace() for c in line) # 공백 개수

```

결과 출력

```
print(lowercase, uppercase, digits, whitespaces)
except EOFError:
```

```
    break # 더 이상 입력이 없으면 종료
```

이 코드는 다음과 같이 작동합니다:

무한 반복문을 사용하여 사용자로부터 계속해서 입력을 받습니다. 입력의 끝(EOF)에 도달하면 반복문을 종료합니다.

각 줄에 대해 `islower()`, `isupper()`, `isdigit()`, `isspace()` 메소드를 사용하여 소문자, 대문자, 숫자, 공백의 개수를 세어 합산합니다. 이를 위해 컴프리헨션(`comprehension`)과 `sum()` 함수를 사용합니다.

계산된 각 카테고리의 개수를 공백으로 구분하여 출력합니다.

파이썬에서는 `EOFError` 예외를 사용하여 입력의 끝을 감지합니다. 이는 주로 파일 입력이나 표준 입력이 끝났을 때 발생합니다. 입력을 모두 처리한 후에는 자동으로 반복문에서 빠져나와 프로그램을 종료합니다.

문제4) 원소들의 곱과 합 비교

정수가 담긴 리스트 num_list가 주어질 때, 모든 원소들의 곱이 모든 원소들의 합의 제곱보다 작으면 1을 크면 0을 return하도록 solution 함수를 완성해주세요.

제한사항

$2 \leq \text{num_list의 길이} \leq 10$

$1 \leq \text{num_list의 원소} \leq 9$

입력 : 첫째 줄부터 N번째 줄까지 문자열이 주어진다. ($1 \leq N \leq 100$) 문자열의 길이는 100을 넘지 않는다.

출력 : 첫째 줄부터 N번째 줄까지 각각의 문자열에 대해서 소문자, 대문자, 숫자, 공백의 개수를 공백으로 구분해 출력한다.

입출력 예 설명

예제 입력 1 : [3, 4, 5, 2, 1]

예제 출력 1 : 1

모든 원소의 곱은 120, 합의 제곱은 225이므로 1을 return합니다.

예제 입력 2 : [5, 7, 8, 3]

예제 출력 2 : 0

모든 원소의 곱은 840, 합의 제곱은 529이므로 0을 return합니다.

이 문제를 해결하기 위해 Java 프로그램을 작성할 때, 주어진 리스트(num_list)의 모든 원소들의 곱과 합의 제곱을 계산하여 비교해야 합니다. 문제의 제한사항을 고려하며 코드를 작성해보겠습니다.

원소들의 곱 계산하기: 리스트의 모든 원소를 순회하며 곱을 계산합니다. 초기 곱의 값은 1로 시작해야 합니다(곱셈의 항등원).

원소들의 합 계산하기: 동일한 방식으로 모든 원소의 합을 계산합니다. 초기 합의 값은 0으로 시작합니다(덧셈의 항등원).

합의 제곱 계산하기: 모든 원소의 합을 구한 후, 이 값을 제곱합니다.

곱과 합의 제곱 비교하기: 원소들의 곱이 합의 제곱보다 작은지 확인하여, 작으면 1을, 크면 0을 반환합니다.

자바로 구현한 예제 코드는 다음과 같습니다:

```
public class Solution {
    public int solution(int[] num_list) {
        // 원소들의 곱과 합을 초기화합니다.
        int product = 1;
        int sum = 0;

        // 리스트의 모든 원소를 순회하며 곱과 합을 계산합니다.
        for (int num : num_list) {
            product *= num;
            sum += num;
        }
    }
}
```

```

    }

    // 합의 제곱을 계산합니다.
    int sumSquared = sum * sum;

    // 원소들의 곱이 합의 제곱보다 작은지 비교하여 결과를 반환합니다.
    if (product < sumSquared) {
        return 1;
    } else {
        return 0;
    }
}

public static void main(String[] args) {
    Solution sol = new Solution();

    // 예제 입력 1
    int[] numList1 = {3, 4, 5, 2, 1};
    System.out.println(sol.solution(numList1)); // 예제 출력 1: 1

    // 예제 입력 2
    int[] numList2 = {5, 7, 8, 3};
    System.out.println(sol.solution(numList2)); // 예제 출력 2: 0
}
}

```

이 코드는 다음과 같은 과정을 거칩니다:

solution 메서드는 정수 배열 num_list를 입력으로 받습니다.

배열의 모든 원소에 대해 곱(product)과 합(sum)을 계산합니다.

계산된 합을 제곱하여 sumSquared에 저장합니다.

product가 sumSquared보다 작으면 1을, 그렇지 않으면 0을 반환합니다.

위 코드는 주어진 문제의 요구사항을 충족시키며, 예제 입력에 대한 올바른 출력을 생성할 것입니다.

Python으로 이 문제를 해결하는 방법은 Java와 매우 유사합니다. 주요 차이점은 Python의 문법과 표현 방식입니다. Python에서는 루프를 사용하여 각 원소를 순회하며 곱과 합을 계산할 수 있고, 리스트의 원소에 대한 연산을 좀 더 간결하게 작성할 수 있습니다.

```
def solution(num_list):
```

```
    # 원소들의 곱과 합을 초기화합니다.
```

```
    product = 1
```

```
    sum = 0
```

```
# 리스트의 모든 원소를 순회하며 곱과 합을 계산합니다.  
for num in num_list:  
    product *= num  
    sum += num  
  
# 합의 제곱을 계산합니다.  
sum_squared = sum ** 2  
  
# 원소들의 곱이 합의 제곱보다 작은지 비교하여 결과를 반환합니다.  
return 1 if product < sum_squared else 0
```

예제 입력 1

```
num_list1 = [3, 4, 5, 2, 1]  
print(solution(num_list1)) # 예제 출력 1: 1
```

예제 입력 2

```
num_list2 = [5, 7, 8, 3]  
print(solution(num_list2)) # 예제 출력 2: 0
```

이 Python 코드는 주어진 문제의 로직을 그대로 따릅니다:

solution 함수는 정수 리스트 num_list를 매개변수로 받습니다.

각 원소에 대해 곱(product)과 합(sum)을 계산합니다.

합을 제곱하여 sum_squared에 저장합니다.

product가 sum_squared보다 작은 경우 1을, 그렇지 않은 경우 0을 반환합니다.

Python의 ** 연산자는 제곱을 계산하는 데 사용됩니다. 이 코드 역시 주어진 입력에 대해 정확한 결과를 출력할 것입니다.

문제5) 주사위 게임

1부터 6까지 숫자가 적힌 주사위가 세 개 있습니다. 세 주사위를 굴렀을 때 나온 숫자를 각각 a, b, c라고 했을 때 얻는 점수는 다음과 같습니다.

세 숫자가 모두 다르다면 $a + b + c$ 점을 얻습니다.

세 숫자 중 어느 두 숫자는 같고 나머지 다른 숫자는 다르다면 $(a + b + c) \times (a\text{제공} + b\text{제공} + c\text{제공})$ 점을 얻습니다.

세 숫자가 모두 같다면 $(a + b + c) \times (a\text{제공} + b\text{제공} + c\text{제공}) \times (a\text{세제공} + b\text{세제공} + c\text{세제공})$ 점을 얻습니다.

세 정수 a, b, c가 매개변수로 주어질 때, 얻는 점수를 return 하는 함수를 작성하세요.

제한사항 : a, b, c는 1이상 6이하의 정수입니다.

입출력 예

a	b	c	result
2	6	1	9
5	3	3	473
4	4	4	110592

이 문제를 해결하기 위해 자바로 작성하는 함수는 주어진 세 숫자 a, b, c의 관계를 판별하여 해당하는 규칙에 따라 점수를 계산해야 합니다. 접근 방식은 다음과 같습니다:

세 숫자가 모두 같은 경우를 확인합니다. 이 경우는 가장 먼저 확인해야 합니다. 왜냐하면 세 숫자가 같다는 조건이 가장 구체적이기 때문입니다.

세 숫자 중 어느 두 숫자가 같은 경우를 확인합니다. 이 경우, 나머지 한 숫자는 다릅니다.

세 숫자가 모두 다른 경우, 기본 규칙을 적용합니다.

각 조건에 따른 점수 계산식은 문제에서 제공된 대로 적용됩니다.

자바로 구현한 코드는 다음과 같습니다:

```
public class DiceGame {
    public static int calculateScore(int a, int b, int c) {
        if (a == b && b == c) {
            // 세 숫자가 모두 같은 경우
            return (a + b + c) * (a*a + b*b + c*c) * (a*a*a + b*b*b + c*c*c);
        } else if (a == b || a == c || b == c) {
            // 세 숫자 중 어느 두 숫자가 같은 경우
            return (a + b + c) * (a*a + b*b + c*c);
        } else {
            // 세 숫자가 모두 다른 경우
            return a + b + c;
        }
    }
}
```

```

    }

    public static void main(String[] args) {
        System.out.println(calculateScore(2, 6, 1)); // 9
        System.out.println(calculateScore(5, 3, 3)); // 473
        System.out.println(calculateScore(4, 4, 4)); // 110592
    }
}

```

이 코드는 다음과 같이 동작합니다:

calculateScore 메서드는 세 개의 정수 a, b, c를 매개변수로 받아들입니다.

세 숫자가 모두 같다면, 첫 번째 조건에 따라 계산식을 적용합니다.

세 숫자 중 어느 두 숫자가 같다면, 두 번째 조건에 따라 계산식을 적용합니다.

그 외의 경우(세 숫자가 모두 다른 경우), 세 숫자의 합을 그대로 반환합니다.

이 로직은 문제에서 제시된 조건과 일치하며, 주어진 예시 입력에 대해 올바른 결과를 출력합니다.

Python으로 이 문제를 풀 때, 자바 소스 코드의 로직을 그대로 따르되, Python의 문법과 특성을 활용하여 코드를 작성할 수 있습니다. 아래는 자바 코드를 Python으로 변환한 예제입니다:

```

def calculate_score(a, b, c):
    if a == b == c:
        # 세 숫자가 모두 같은 경우
        return (a + b + c) * (a**2 + b**2 + c**2) * (a**3 + b**3 + c**3)
    elif a == b or a == c or b == c:
        # 세 숫자 중 어느 두 숫자가 같은 경우
        return (a + b + c) * (a**2 + b**2 + c**2)
    else:
        return a + b + c # 세 숫자가 모두 다른 경우

```

예제 입력과 출력을 테스트합니다.

```

print(calculate_score(2, 6, 1)) # 출력: 9
print(calculate_score(5, 3, 3)) # 출력: 473
print(calculate_score(4, 4, 4)) # 출력: 110592

```

이 Python 코드는 다음과 같은 방식으로 작동합니다. calculate_score 함수는 세 개의 정수 a, b, c를 인자로 받습니다. 첫 번째 조건문에서 세 숫자가 모두 같은지 확인합니다. Python에서는 a == b == c와 같은 연속적인 비교가 가능합니다. elif를 사용하여 세 숫자 중 어느 두 숫자가 같은지 확인합니다. 이 경우, 나머지 조건(a == b or a == c or b == c)이 참이면 해당 계산을 수행합니다.

만약 위의 두 조건에 모두 해당하지 않는다면, 세 숫자가 모두 다른 경우로 간주하고 a + b + c를 반환합니다. Python의 ** 연산자는 제곱을 계산하는 데 사용되며, 자바 코드에서의 Math.pow() 함수의 역할을 대신합니다. 이 코드 역시 문제의 요구사항을 충족하며 주어진 예제 입력에 대한 올바른 결과를 제공합니다.

문제6) 두 정수 사이의 합

두 정수 a, b가 주어졌을 때 a와 b 사이에 속한 모든 정수의 합을 리턴하는 함수, solution을 완성하세요.

예를 들어 a = 3, b = 5인 경우, $3 + 4 + 5 = 12$ 이므로 12를 리턴합니다.

제한 조건

a와 b가 같은 경우는 둘 중 아무 수나 반환하세요.

a와 b는 -10,000,000 이상 10,000,000 이하인 정수입니다.

a와 b의 대소관계는 정해져 있지 않습니다.

입출력 예

a	b	return
3	5	12
3	3	3
5	3	12

이 문제를 해결하기 위한 자바 코드는 두 정수 a와 b 사이의 모든 정수를 포함하여 그 합을 계산해야 합니다. 이를 위해 두 가지 주요 접근 방식이 있습니다:

루프 사용: a와 b 사이의 모든 정수를 순회하며 합을 구합니다.

수학적 공식 사용: 등차수열의 합 공식을 사용하여 계산합니다. 이 방식이 더 효율적입니다.

두 번째 방식인 등차수열의 합 공식을 사용하는 방법으로 문제를 해결해 보겠습니다.

등차수열의 합 공식은 다음과 같습니다. $\text{합} = n(a_1 + a_n) / 2$

여기서 n은 항의 수, a_1 은 첫 번째 항, a_n 은 마지막 항입니다. 두 정수 사이의 정수의 개수는 $|a-b|+1$ 이고, 첫 번째 항과 마지막 항의 합은 $a+b$ 입니다.

자바로 구현하면 다음과 같습니다:

```
public class Solution {
    public long solution(int a, int b) {
        // a와 b가 같은 경우, a 또는 b를 리턴합니다.
        if (a == b) {
            return a;
        }

        // a와 b의 대소관계를 정하지 않고, 두 수 사이의 모든 수의 합을 구합니다.
        // long 타입을 사용하여 오버플로우를 방지합니다.
        long total = 0;

        // a와 b 중에서 작은 값을 start, 큰 값을 end로 합니다.
        int start = Math.min(a, b);
        int end = Math.max(a, b);
```

```

// 등차수열의 합 공식을 사용합니다.
total = (long)(end - start + 1) * (start + end) / 2;

return total;
}

public static void main(String[] args) {
    Solution sol = new Solution();

    System.out.println(sol.solution(3, 5)); // 12
    System.out.println(sol.solution(3, 3)); // 3
    System.out.println(sol.solution(5, 3)); // 12
}
}

```

이 코드는 두 수 a와 b의 대소관계에 상관없이 정확한 합을 계산합니다. Math.min()과 Math.max()를 사용하여 두 수 중 작은 수와 큰 수를 찾고, 이를 통해 등차수열의 합 공식에 필요한 값들을 구합니다. 결과적으로, 이 방법은 모든 입력 조건에 대해 올바른 합을 효율적으로 계산합니다.

Python에서는 자바 코드와 같은 로직을 사용할 수 있지만, 더 간결한 문법으로 문제를 해결할 수 있습니다. 두 정수 사이의 모든 정수의 합을 계산하기 위해 등차수열의 합 공식을 사용하는 방법을 그대로 적용할 수 있습니다.

Python으로 변환된 코드는 다음과 같습니다:

```

def solution(a, b):
    # a와 b가 같은 경우, a 또는 b를 바로 리턴합니다.
    if a == b:
        return a

    # a와 b 중에서 작은 값을 start, 큰 값을 end로 설정합니다.
    start = min(a, b)
    end = max(a, b)

    # 등차수열의 합 공식을 사용합니다.
    total = (end - start + 1) * (start + end) // 2

    return total

# 테스트 코드
print(solution(3, 5)) # 12
print(solution(3, 3)) # 3
print(solution(5, 3)) # 12

```

이 Python 코드는 자바 코드와 동일한 로직을 따르되, Python의 내장 함수 `min()`과 `max()`를 사용하여 두 정수 사이의 최소값과 최대값을 찾습니다. 그 후, 등차수열의 합 공식을 적용하여 합계를 계산합니다. Python에서는 정수 나눗셈을 위해 `//` 연산자를 사용하여 결과가 항상 정수임을 보장합니다. 이 코드 역시 모든 입력 조건에 대해 올바른 결과를 제공합니다.

문제7) 행렬의 덧셈 구하기

행렬의 덧셈은 행과 열의 크기가 같은 두 행렬의 같은 행, 같은 열의 값을 서로 더한 결과가 됩니다. 2개의 행렬 arr1과 arr2를 입력받아, 행렬 덧셈의 결과를 반환하는 함수, solution을 완성해주세요.

제한 조건 : 행렬 arr1, arr2의 행과 열의 길이는 500을 넘지 않습니다.

입출력 예

arr1	arr2	return
[[1,2],[2,3]]	[[3,4],[5,6]]	[[4,6],[7,9]]
[[1],[2]]	[[3],[4]]	[[4],[6]]

이 문제를 해결하기 위해 자바에서 행렬 덧셈을 수행하는 함수를 구현해보겠습니다. 행렬은 2차원 배열로 표현되며, 두 행렬의 동일한 위치에 있는 원소끼리 더하여 새로운 행렬을 생성합니다. 이 과정은 두 개의 중첩된 for 루프를 사용하여 각 원소에 접근하고 계산하는 방식으로 수행할 수 있습니다.

아래의 코드는 두 행렬의 덧셈을 수행하는 solution 함수를 구현한 것입니다:

```
public class Solution {
    public int[][] solution(int[][] arr1, int[][] arr2) {
        // 두 행렬의 크기는 동일하다고 가정합니다.
        int rows = arr1.length; // 행의 개수
        int cols = arr1[0].length; // 열의 개수

        // 결과 행렬을 저장할 새로운 2차원 배열을 초기화합니다.
        int[][] result = new int[rows][cols];

        // 두 행렬의 동일한 위치에 있는 원소들을 더하여 결과 행렬에 저장합니다.
        for (int i = 0; i < rows; i++) {
            for (int j = 0; j < cols; j++) {
                result[i][j] = arr1[i][j] + arr2[i][j];
            }
        }

        // 계산된 결과 행렬을 반환합니다.
        return result;
    }

    public static void main(String[] args) {
        Solution sol = new Solution();

        // 테스트 케이스 1
        int[][] arr1 = {{1, 2}, {2, 3}};
```

```

int[][] arr2 = {{3, 4}, {5, 6}};
int[][] result = sol.solution(arr1, arr2);

// 결과 출력
for (int[] row : result) {
    for (int value : row) {
        System.out.print(value + " ");
    }
    System.out.println();
}

// 테스트 케이스 2
arr1 = new int[][]{{1}, {2}};
arr2 = new int[][]{{3}, {4}};
result = sol.solution(arr1, arr2);

// 결과 출력
for (int[] row : result) {
    for (int value : row) {
        System.out.print(value + " ");
    }
    System.out.println();
}
}
}

```

이 코드는 두 행렬 arr1과 arr2를 인자로 받아, 두 행렬의 같은 위치에 있는 원소들을 더한 결과를 result라는 새로운 행렬에 저장합니다. 그 후, 계산된 result 행렬을 반환합니다. main 메서드에서는 두 가지 테스트 케이스를 실행하여 결과를 출력하고, 함수가 올바르게 동작하는지 확인합니다.

Python에서 행렬 덧셈을 구현할 때는 리스트 컴프리헨션을 활용하여 코드를 더 간결하게 만들 수 있습니다. 아래는 자바 코드를 Python으로 변환한 예시입니다:

```

def solution(arr1, arr2):
    # 리스트 컴프리헨션을 사용하여 두 행렬의 동일한 위치의 원소들을 더함
    result = [[c1 + c2 for c1, c2 in zip(r1, r2)] for r1, r2 in zip(arr1, arr2)]
    return result

# 테스트 케이스 1
arr1 = [[1, 2], [2, 3]]
arr2 = [[3, 4], [5, 6]]
result = solution(arr1, arr2)

```

```
# 결과 출력
```

```
for row in result:  
    print(row)
```

```
# 테스트 케이스 2
```

```
arr1 = [[1], [2]]
```

```
arr2 = [[3], [4]]
```

```
result = solution(arr1, arr2)
```

```
# 결과 출력
```

```
for row in result:  
    print(row)
```

이 코드는 두 행렬 arr1과 arr2의 각 행을 zip 함수로 결합하고, 그 결과를 다시 zip하여 각 위치의 원소를 더하는 방식으로 작동합니다. zip 함수는 여러 컨테이너(리스트, 튜플 등)의 아이템들을 동시에 접근할 수 있게 해 줍니다. 이 방식은 코드를 매우 직관적이고 간결하게 만들어 줍니다.

결과적으로, 이 Python 코드는 자바 코드와 동일한 작업을 수행하지만, 보다 Pythonic한 방식으로 구현되어 있습니다.

문제8) 구명보트를 이용하여 구출

무인도에 갇힌 사람들을 구명보트를 이용하여 구출하려고 합니다. 구명보트는 작아서 한 번에 최대 2명씩 밖에 탈 수 없고, 무게 제한도 있습니다.

예를 들어, 사람들의 몸무게가 [70kg, 50kg, 80kg, 50kg]이고 구명보트의 무게 제한이 100kg이라면 2번째 사람과 4번째 사람은 같이 탈 수 있지만 1번째 사람과 3번째 사람의 무게의 합은 150kg이므로 구명보트의 무게 제한을 초과하여 같이 탈 수 없습니다.

구명보트를 최대한 적게 사용하여 모든 사람을 구출하려고 합니다.

사람들의 몸무게를 담은 배열 `people`과 구명보트의 무게 제한 `limit`가 매개변수로 주어질 때, 모든 사람을 구출하기 위해 필요한 구명보트 개수의 최솟값을 `return` 하도록 `solution` 함수를 작성해주세요.

제한사항

- 무인도에 갇힌 사람은 1명 이상 50,000명 이하입니다.
- 각 사람의 몸무게는 40kg 이상 240kg 이하입니다.
- 구명보트의 무게 제한은 40kg 이상 240kg 이하입니다.
- 구명보트의 무게 제한은 항상 사람들의 몸무게 중 최댓값보다 크게 주어지므로 사람들을 구출할 수 없는 경우는 없습니다.

입출력 예

people	limit	return
[70, 50, 80, 50]	100	3
[70, 80, 50]	100	3

이 문제는 효율적으로 구명보트를 사용하여 사람들을 구출하는 최적의 방법을 찾는 것입니다. 주어진 제한사항을 고려할 때, 먼저 사람들의 몸무게 배열을 정렬하여 가장 가벼운 사람과 가장 무거운 사람을 함께 태울 수 있는지 판단하는 것이 핵심 아이디어입니다. 만약 둘의 무게 합이 구명보트의 제한 무게 이하라면 함께 태우고, 그렇지 않다면 가장 무거운 사람을 혼자 태워 보냅니다. 이 과정을 모든 사람이 구출될 때까지 반복합니다.

자바로 이 문제를 해결하는 방법은 다음과 같습니다:

```
import java.util.Arrays;

public class Solution {
    public int solution(int[] people, int limit) {
        // 사람들의 몸무게 배열을 오름차순으로 정렬합니다.
        Arrays.sort(people);

        // 필요한 구명보트의 개수를 저장할 변수를 초기화합니다.
        int answer = 0;

        // 가장 가벼운 사람의 인덱스
        int i = 0;
```

```

// 가장 무거운 사람의 인덱스
int j = people.length - 1;

// 가벼운 사람과 무거운 사람을 두 포인터로 접근합니다.
while (i <= j) {
    // 두 사람의 무게 합이 제한 무게 이하인 경우 함께 태울 수 있습니다.
    if (people[i] + people[j] <= limit) {
        // 가벼운 사람의 인덱스를 오른쪽으로 이동합니다.
        i++;
    }
    // 무거운 사람은 항상 구명보트에 탑니다.
    j--;
    // 구명보트의 개수를 증가시킵니다.
    answer++;
}

return answer;
}

public static void main(String[] args) {
    Solution sol = new Solution();
    System.out.println(sol.solution(new int[]{70, 50, 80, 50}, 100)); // 3
    System.out.println(sol.solution(new int[]{70, 80, 50}, 100)); // 3
}
}

```

이 코드는 다음과 같이 작동합니다.

사람들의 몸무게 배열을 오름차순으로 정렬합니다.

두 포인터를 사용하여 배열의 양 끝에서부터 중간으로 접근합니다. 하나는 가장 가벼운 사람을 가리키고 다른 하나는 가장 무거운 사람을 가리킵니다.

가장 가벼운 사람과 가장 무거운 사람의 몸무게 합이 구명보트의 제한 무게 이하인 경우, 두 사람 모두를 함께 태울 수 있습니다. 이 경우, 두 포인터 모두를 이동시킵니다.

몸무게 합이 제한 무게를 초과하는 경우, 가장 무거운 사람만 구명보트에 태워 보냅니다. 이 경우, 무거운 사람을 가리키는 포인터만 이동시킵니다.

필요한 구명보트의 개수를 모두 합하여 반환합니다.

이 알고리즘은 모든 사람을 구출하기 위해 필요한 구명보트의 최소 개수를 효율적으로 계산합니다.

Python에서는 자바 코드의 로직을 유지하면서, Python의 문법을 활용해 보다 간결하게 문제를 해결할 수 있습니다. 주어진 문제를 푸는 Python 코드는 다음과 같습니다:

```

def solution(people, limit):
    # 사람들의 몸무게를 오름차순으로 정렬

```

```

people.sort()

# 필요한 구명보트의 개수
answer = 0

# 가장 가벼운 사람의 인덱스
i = 0
# 가장 무거운 사람의 인덱스
j = len(people) - 1

# 가벼운 사람과 무거운 사람을 투 포인터 방식으로 접근
while i <= j:
    # 두 사람의 무게 합이 제한 무게 이하인 경우 함께 태울 수 있음
    if people[i] + people[j] <= limit:
        i += 1

    j -= 1 # 무거운 사람은 항상 구명보트에 탑승
    answer += 1 # 구명보트의 개수 증가

return answer

# 테스트 케이스
print(solution([70, 50, 80, 50], 100)) # 3
print(solution([70, 80, 50], 100)) # 3

```

이 코드는 자바 코드와 동일한 알고리즘을 따릅니다:

사람들의 몸무게를 오름차순으로 정렬합니다.

가장 가벼운 사람과 가장 무거운 사람을 각각 가리키는 두 개의 포인터(i, j)를 사용합니다.

i 와 j 가 만나거나 교차할 때까지 루프를 실행하며, 조건에 따라 사람들을 구명보트에 태우고 포인터를 조정합니다.

필요한 구명보트의 최소 개수를 계산하여 반환합니다.

Python의 리스트 정렬(`sort()`)과 인덱스 조작을 통해 간결하고 효율적으로 문제를 해결할 수 있습니다.

문제9) 숫자 맞추기 게임

숫자 맞추기 게임은 초등학교 학생들 사이에서 유행하는 게임이다.

이 게임을 시작할 때는 친구가 숫자 하나를 머릿속에 생각해야 한다. 이 숫자를 n_0 이라고 하자. 그리고 나서 다음과 같이 게임을 진행한다.

친구에게 $n_1 = 3 \times n_0$ 계산을 하라고 한 뒤, n_1 이 짝수인지 홀수인지를 말해달라고 한다.

n_1 이 짝수라면, $n_2 = n_1/2$ 를, 홀수라면 $n_2 = (n_1+1)/2$ 를 계산해달라고 한다.

$n_3 = 3 \times n_2$ 의 계산을 부탁한다.

친구에게 $n_4 = n_3/9$ 를 계산한 뒤, 그 값을 말해달라고 한다. (n_4 는 나눗셈의 몫이다)

자 이제, n_1 이 짝수였다면, $n_0 = 2 \times n_4$ 로, 홀수였다면, $n_0 = 2 \times n_4 + 1$ 로 처음 친구가 생각한 숫자를 맞출 수 있다.

예를 들어, 친구가 생각한 수가 $n_0=37$ 이었다면, $n_1 = 111$ 이 되고 홀수이다. 그 다음 $n_2 = 56$, $n_3 = 168$, $n_4 = 18$ 이 된다. 친구는 n_4 를 알려주게 된다.

그럼 $2 \times n_4 + 1 = 37$ 이기 때문에, 친구가 제일 처음 생각한 숫자를 맞출 수 있다.

n_0 이 주어졌을 때, n_1 이 홀수인지 짝수인지와 n_4 를 구하는 프로그램을 작성하시오.

입력 : 입력은 여러 개의 테스트 케이스로 이루어져 있다. 각 테스트 케이스는 한 줄로 이루어져 있고, n_0 으로 이루어져 있다. ($0 < n_0 < 1,000,000$) 입력의 마지막 줄에는 0이 하나 주어진다.

출력 : 각 테스트 케이스에 대해서, 케이스 번호를 출력하고 n_1 이 짝수라면 'even', 홀수라면 'odd'를 출력하고, n_4 를 출력한다.

예제 입력 1

37

38

0

예제 출력 1

1. odd 18

2. even 19

이 문제를 해결하기 위해 자바 프로그램을 작성할 때, 주어진 단계에 따라 각 단계별로 계산을 수행하고, 필요한 정보(짝수/홀수, n_4 의 값)를 출력해야 합니다. 각 단계를 따라가며 계산 과정을 구현하는 것이 핵심입니다. 자바로 구현한 코드 예제는 다음과 같습니다:

```
import java.util.Scanner;

public class NumberGuessingGame {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        int testCase = 1; // 테스트 케이스 번호를 1부터 시작

        while (true) {
            int n0 = scanner.nextInt(); // n0 입력 받기
```

```

        // 입력된 n0가 0이면 반복문 종료
        if (n0 == 0) break;

        // 게임의 단계에 따라 계산 수행
        int n1 = 3 * n0;
        int n2 = n1 % 2 == 0 ? n1 / 2 : (n1 + 1) / 2;
        int n3 = 3 * n2;
        int n4 = n3 / 9;

        // n1이 짝수인지 홀수인지에 따라 출력
        String parity = n1 % 2 == 0 ? "even" : "odd";

        System.out.printf("%d. %s %d\n", testCase, parity, n4);
        testCase++; // 테스트 케이스 번호 증가
    }

    scanner.close();
}
}

```

이 코드는 다음과 같이 작동합니다:

Scanner 클래스를 사용하여 표준 입력으로부터 n0 값을 입력 받습니다.

입력된 n0가 0이면 프로그램을 종료합니다. 0은 입력의 끝을 나타냅니다.

주어진 규칙에 따라 n1, n2, n3, 그리고 n4를 차례대로 계산합니다.

n1이 짝수인지 홀수인지를 판별하여 even 또는 odd를 출력하고, n4의 값을 출력합니다.

테스트 케이스 번호를 1씩 증가시키며 반복합니다.

이 프로그램은 주어진 예제 입력에 대해 올바른 예제 출력을 생성합니다.

Python에서는 같은 로직을 사용하여 이 문제를 풀 수 있으며, 입력을 처리하기 위해 input() 함수를 사용하고, 반복문과 조건문을 통해 계산 과정을 구현할 수 있습니다. 여기서는 무한 루프 내에서 입력을 받고, 입력된 값이 0일 때 루프를 종료하는 방식으로 작성해 보겠습니다.

```
test_case = 1
```

```
while True:
```

```
    n0 = int(input()) # n0 입력 받기
```

```
    if n0 == 0:      # 입력된 n0가 0이면 루프 종료
```

```
        break
```

```
    # 게임의 단계에 따라 계산 수행
```

```
    n1 = 3 * n0
```



```
n2 = n1 // 2 if n1 % 2 == 0 else (n1 + 1) // 2
n3 = 3 * n2
n4 = n3 // 9

# n1이 짝수인지 홀수인지에 따라 출력
parity = "even" if n1 % 2 == 0 else "odd"

print(f"{test_case}. {parity} {n4}")
test_case += 1 # 테스트 케이스 번호 증가
```

이 코드는 자바 버전과 유사하게 작동합니다:

무한 루프를 사용하여 사용자로부터 `n0` 값을 입력 받습니다. `input()` 함수는 문자열을 반환하므로, `int()`를 사용하여 정수로 변환합니다.

입력된 `n0`가 0이면 `break` 문을 사용하여 루프를 종료합니다.

`n1`, `n2`, `n3`, `n4`를 순서대로 계산합니다. Python에서 정수 나눗셈을 위해 `//` 연산자를 사용합니다.

`n1`의 홀수/짝수 여부에 따라 "even" 또는 "odd" 문자열을 결정하고, 테스트 케이스 번호와 함께 출력합니다.

테스트 케이스 번호를 1씩 증가시키며, 사용자가 0을 입력할 때까지 계속 진행합니다.

문제10) 좋은 수열 출력

숫자 1, 2, 3으로만 이루어지는 수열이 있다. 임의의 길이의 인접한 두 개의 부분 수열이 동일한 것이 있으면, 그 수열을 나쁜 수열이라고 부른다. 그렇지 않은 수열은 좋은 수열이다.

다음은 나쁜 수열의 예이다.

33

32121323

123123213

다음은 좋은 수열의 예이다.

2

32

32123

1232123

길이가 N인 좋은 수열들을 N자리의 정수로 보아 그중 가장 작은 수를 나타내는 수열을 구하는 프로그램을 작성하라. 예를 들면, 1213121과 2123212는 모두 좋은 수열이지만 그 중에서 작은 수를 나타내는 수열은 1213121이다.

입력 : 입력은 숫자 N하나로 이루어진다. N은 1 이상 80 이하이다.

출력 : 첫 번째 줄에 1, 2, 3으로만 이루어져 있는 길이가 N인 좋은 수열들 중에서 가장 작은 수를 나타내는 수열만 출력한다. 수열을 이루는 1, 2, 3들 사이에는 빈칸을 두지 않는다.

예제 입력 1 : 7

예제 출력 1 : 1213121

이 문제를 해결하기 위해서는 재귀 함수를 사용하여 모든 가능한 수열을 생성하고, 생성 과정에서 나쁜 수열이 되는 경우를 배제하는 방식으로 접근할 수 있습니다. 길이가 N인 좋은 수열 중 가장 작은 수를 찾기 위해, 우선적으로 1, 2, 3의 순서대로 수열에 추가해 보고, 각 단계에서 수열이 좋은 수열인지 확인합니다. 만약 현재까지의 수열이 나쁜 수열의 조건을 만족한다면, 그 수열을 더 이상 확장하지 않고 다른 경로를 탐색합니다.

자바로 구현한 코드 예제는 다음과 같습니다:

```
import java.util.Scanner;

public class GoodSequence {
    private static int N;
    private static boolean finished = false;

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        N = scanner.nextInt();
        findGoodSequence("");
        scanner.close();
    }
}
```

```

}

private static void findGoodSequence(String sequence) {
    if (finished) return;

    if (sequence.length() == N) {
        System.out.println(sequence);
        finished = true;
        return;
    }

    for (int i = 1; i <= 3; i++) {
        if (isGoodSequence(sequence + i)) {
            findGoodSequence(sequence + i);
        }
    }
}

private static boolean isGoodSequence(String sequence) {
    int len = sequence.length();
    for (int i = 1; i <= len / 2; i++) {
        if (sequence.substring(len - i * 2, len - i).equals(sequence.substring(len - i, len))) {
            return false;
        }
    }
    return true;
}
}

```

이 코드는 다음과 같이 동작합니다:

사용자로부터 수열의 길이 N을 입력 받습니다.

빈 문자열에서 시작하여 좋은 수열을 찾는 findGoodSequence 함수를 호출합니다.

findGoodSequence 함수는 재귀적으로 수열을 생성합니다. 각 단계에서 1, 2, 3을 현재 수열에 추가하고, isGoodSequence 함수를 호출하여 현재까지의 수열이 좋은 수열인지 검사합니다.

만약 수열이 나쁜 수열이면, 그 수열은 더 이상 고려하지 않고 다른 숫자를 추가한 수열을 검사합니다.

길이가 N인 좋은 수열을 찾으면 출력하고 탐색을 종료합니다.

isGoodSequence 함수는 주어진 수열이 좋은 수열인지를 판별합니다. 수열의 각 부분을 비교하여 나쁜 수열의 조건(인접한 두 부분 수열이 동일한 경우)을 만족하는지 검사합니다.

이 방법으로 길이가 N인 좋은 수열 중 가장 작은 수를 나타내는 수열을 찾을 수 있습니다.

Python으로 이 문제를 해결하기 위한 접근 방법은 자바 코드와 유사합니다. 재귀 함수를 사용하여 모든 가능한 수열을 생성하면서, 현재까지의 수열이 좋은 수열인지를 검사하고, 나쁜 수열을 배제하는 방식으로 길이가

N인 좋은 수열 중 가장 작은 수를 찾습니다. Python의 문자열 처리 능력을 활용하여 이 과정을 구현할 수 있습니다.

```
def find_good_sequence(sequence, n):
    if len(sequence) == n:
        print(sequence)
        # 재귀 호출 중단을 위해 프로그램 전체를 종료
        exit()
    for i in '123':
        if is_good_sequence(sequence + i):
            find_good_sequence(sequence + i, n)

def is_good_sequence(sequence):
    length = len(sequence)
    for i in range(1, length // 2 + 1):
        # 현재 수열의 마지막 부분과 그 직전 부분을 비교
        if sequence[-i:] == sequence[-2*i:-i]:
            return False
    return True

if __name__ == "__main__":
    n = int(input())
    find_good_sequence("", n)
```

이 코드는 다음과 같이 작동합니다:

길이 n을 입력 받아 find_good_sequence 함수를 빈 문자열과 함께 호출하여 시작합니다.

find_good_sequence 함수는 재귀적으로 수열을 생성합니다. 1, 2, 3 각각을 현재 수열에 추가하고, 추가된 수열이 좋은 수열인지 is_good_sequence 함수를 통해 검사합니다.

수열이 나쁜 수열이면 그 경로는 더 이상 탐색하지 않고 다른 경로를 탐색합니다.

길이가 n인 좋은 수열을 찾으면 그 수열을 출력하고 프로그램을 종료합니다.

is_good_sequence 함수는 주어진 수열이 좋은 수열인지 검사합니다. 수열의 끝 부분과 그 앞부분을 비교하여 인접한 두 부분 수열이 동일한지 확인합니다.

이 알고리즘은 길이가 n인 좋은 수열 중 가장 작은 수를 나타내는 수열을 효율적으로 찾아 출력합니다.