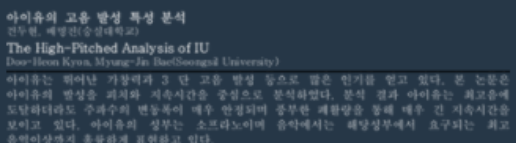
4단 고음

문제 설명

4단 고음


```
I'm in my dream~♪ ~♪ ~♪
```

IU는 본인의 장기인 3단 고음으로 유명하다. 그러던 그녀가 어느 날 4단 고음을 성공했고 그녀의 고음은 학계에서 연구가 될 만큼 유명해졌다 [1].


아이유의 고음 발성 특성 분석
The High-Pitched Analysis of IU

[1] 견두헌, 배명진. "아이유의 고음 발성 특성 분석", 한국음향학회, 2011년 춘계학술대회 학술발표논문지

폭포 밑 득음 수련을 하던 어느 날, 그녀는 4단 고음이 끝이 아님을 깨달았다. 3단 고음 직후 3단 고음을 연이어하거나, 3단 고음 중 다시 3단 고음을 해서 음높이를 올리는 방법이다. 어떤 순서로 3단 고음을 했는지에 따라 최종 음높이가 달라지기 때문에, 연속 3단 고음을 연습할 때마다 그 결과를 기록으로 남기기로 했다.

3단 고음은 다음과 같이 적용된다. 1단계에서는 음높이가 세 배가 되며, 2단계와 3단계에서 음높이가 각각 1씩 증가한다. 이를 기록으로 남길 때 * 와 + 기호를 사용하기로 했다. 즉, 3단 고음을 한 번 한 경우는 문자열로 나타내면 다음과 같다.

*++

이때 3단 고음을 마치고 연달아 3단 고음을 한 경우는 *++ `*++` 와 같이 표현할 수 있다. 3단 고음의 2단계를 마친 후 3단 고음을 새로 시작한 다음, 나머지 단계를 이어서 하는 경우는 *+ `*++` +로 표현할 수 있다. (강조된 부분이 2번째 3단 고음을 부른 부분이다.)

이와 같이 * 와 + 로 구성된 문자열이 3단 고음의 규칙을 적용하여 만들 수 있는 문자열인 경우 '올바른 문자열'이라고 하자. 다음의 문자열은 3단 고음의 규칙으로 만들 수 있는 문자열이 아니므로 올바른 문자열이 아니다.

- +**+++
- *+++*+

올바른 문자열에 대해 음높이는 다음과 같이 계산할 수 있다. 시작 음높이는 항상 1이며, 문자열의 처음부터 순서대로 * 기호의 경우 3을 곱하고 + 기호의 경우 1을 더한다. *+*+++ 의 음높이를 계산하는 과정을 예로 들면 아래와 같다.

시작 음 높이: 1

*	+	*	+	+	+
*3	+1	*3	+1	+1	+1

최종 음높이: 15

그날 기분에 따라 최종 음높이를 정하는 IU는 최종 음높이를 결정했을 때 서로 다른 3단 고음 문자열이 몇 가지나 있는지 궁금하다. 여러분의 도움이 필요하다.

입력 형식

- 입력은 `5` 이상 `2^31-1` 이하의 정수 `n` 으로 주어진다.

출력 형식

- 입력을 만족하는 서로 다른 문자열의 수를 리턴한다.

예제 입출력

n	answer
15	1
24	0
41	2
2147483647	1735

```
* ( x 3 ) + ( + 1 )
```

```
* *
```

```
*+ *
```

```
*++*
```

초기 접근

1부터 재귀 형태로 타겟넘버를 목표로 진행하였다.

```
find(int num , int target) {

    if(target == n {
      answer++;
      return;
    }
    find(num * 3 , target);
    find(num + 1 , target);
}
```

=> 모든 경우의 수를 탐색하는 경우이므로 target 까지 도달 할 수는 있지만

target이 15인 경우

1) *+++++++++++++ (15)
2) *****(15)

등등 * 하나당 + 가 두개씩 와야한다는 세부조건을 만족하지 않는 케이스가 나오고

재귀형태기 때문에 쓸데없는 연산이 많아 결국 시간초과가 발생함.

개선 형태

역으로 n에서 시작해서 시작인 1로 돌아가는 경우

n을 1씩 빼주면서 + 카운트를 세고,

n이 3으로 나눠지고, + 가 2번이상 나왔다면, * 가 있다고 생각하고, n을 3으로 나눠준다.

```
    if(ed % 3 == 0 && pCnt >= 2) {
        find(ed / 3, pCnt - 2);
    }
    find(ed - 1, pCnt + 1);
```

만약 n이 3일때 pCnt가 2라면 1로 나눌 수 있기 때문에 조건에 만족할 수 있다.

```
find(int ed, int pCnt) {
    if(ed == 3)
      if(pCnt == 2) {
        answer++;
        return;
      }
}
```

=> 이 경우도 마찬가지로, 정답에 도달 할 수 있지만,
15
ed == 3 일때 까지 하염없이 - 1씩 하면서 도달 했던 pCnt가 2가 아닌 실행때문에 시간 초과

최종 개선

`*` 한개당 `+` 가 두개가 나온다는 사실을 통해 가지치기를 할 필요가 있었다.

`++++++` 현재 이상태라면

`*` 는 3번 나와야한다.

즉 pCnt `/` 2 만큼 나와야 한다는 것.

결국 3 ^ (3) = 27 점은 ed가 넘어줘야 3단 고음을 완성 할 수 있는 것이기 때문에

이 조건을 만족하지 못한다면 음높이를 완성 할 수 없기 때문에 return하여 가지치기한다.

```
if(Math.pow(3, pCnt/2) > ed) {
  return;
}
```

추가 개선 포인트

`=>` 첫 시작은 무조건 `*` 로 시작되고, 마지막은 `++` 로 끝나게 되어있다.

따라서 목표가 15라면

3 < x < 13 의 범위에서 찾기 시작하면 조금이라도 연산을 더 줄 일 수 있다.

코드

```
package Algo_Study_Programmers;

public class Solution_4단고음 {
  static int answer;
  private static void find(int ed, int pCnt) {

    if(Math.pow(3, pCnt/2) > ed) {
      return;
    }
    if(ed == 3) {
      if(pCnt == 2) {
        answer++;
        return;
      }
      return;
    }
    else {
      if(ed % 3 == 0 && pCnt >= 2) {
        find(ed / 3, pCnt - 2);
```

```
        }
        find(ed - 1, pCnt + 1);
      }
    }

    public static int solution(int n) {
        answer = 0;

        find(n, 0);
        return answer;
    }
  public static void main(String[] args) {
    int n = 2147483647;
    System.out.println(solution(n));
  }
}
```