프로그래머스 - 구슬을 나누는 경우의 수

태그

BigInteger

Math.round(factorial(balls) / factorial(balls - share) / factorial(share))

factorial 함수 조합 = n! / (r! * (n-r)!)

```
자바스크립트
구슬을 나누는 경우의 수
문제 설명
머쓱이는 구슬을 친구들에게 나누어주려고 합니다. 구슬은 모두 다르게 생겼습니다. 머쓱이가 갖고 있는 구슬의 개수 balls와 친구들에게 나누어 줄 구슬 개수 share이 매개변수로 주어질 때, balls개의 구슬 중 share개의 구슬을 고르는 가능한 모든 경우의 수를 return 하는 solution 함수를 완성해주세요.

제한사항
1 ≤ balls ≤ 30
1 ≤ share ≤ 30
구슬을 고르는 순서는 고려하지 않습니다.
share ≤ balls

입출력 예 #1
balls: 3  ,share: 2 ,result: 3
서로 다른 구슬 3개 중 2개를 고르는 경우의 수는 3입니다.

입출력 예 #2
balls: 5  ,share: 3 ,result: 10
서로 다른 구슬 5개 중 3개를 고르는 경우의 수는 10입니다.
```

```
const factorial  = (num) => num === 0 ? 1 : num * factorial(num - 1)

function solution(balls, share) {
  return Math.round(factorial(balls) / factorial(balls - share) / factorial(share))
}
```

```
 이것은 factorial 함수를 사용하여 숫자의  factorial을 계산하는 문제를 해결방법입니다.

 factorial 함수는 재귀를 사용하여 숫자의  factorial을 계산합니다. 단일 인수 num을 사용하고 num의 계승을 반환합니다. num이 0이면 1을 반환합니다. 그렇지 않으면 num - 1의  factorial을 곱한 num을 반환합니
```

다.

Solution 함수는 factorial 함수를 사용하여 ball과 share의 조합을 계산합니다. 공의 factorial을 공의 factorial인 공유와 공유의 곱으로 나눔으로써 이를 수행합니다. 마지막으로 Math.round를 사용하여 결과를 반올림하여 반환합니다.

```
function solution(balls, share) {
  let result = 1;
  for (let i = 0; i < share; i++) {
    result = result * (balls - i) / (i + 1);
  }
  return result;
}
```

이 문제를 해결하기 위해 조합 공식을 사용할 수 있습니다. 조합 공식은 항목의 순서가 중요하지 않은 더 큰 세트에서 항목의 하위 집합을 선택하는 방법의 수를 계산하는 데 사용됩니다.

조합 = n! / (r! * (n-r)!)

여기서 n은 구슬(공)의 총 수이고 r은 선택된 구슬(공유)의 수입니다.예를 들어 첫 번째 입력(공: 3, 공유: 2, 결과: 3)에는 총 3개의 구슬이 있고 한 번에 2개를 선택합니다. 이것을 공식에 대입하면 다음과 같습니다.
조합 = 3! / (2! * (3-2)!) = 3 / (2 * 1) = 3/2 = 1.5 우리는 조합의 수에만 관심이 있기 때문에 소수점을 무시하고 정수 값인 1을 취할 수 있습니다.
이 함수는 먼저 변수 결과를 1로 초기화합니다. 그런 다음 공유 횟수를 반복하고 각 반복에서 (balls - i) / (i + 1)을 곱하여 결과를 업데이트합니다. 여기서 i는 현재 변수의 인덱스입니다. 마지막으로 함수는 결과를 반환합니다.

```
import java.lang.*;
import java.math.*;
class Solution {
    public BigInteger solution(int balls, int share) {
        BigInteger n = BigInteger.ONE, m = BigInteger.ONE;
        long select1 = balls-share > share ? balls-share : share;
        long select2 = balls-share > share? share : balls-share;
        for(long i = select1+1; i <= balls; i++)
            n = n.multiply(BigInteger.valueOf(i));
        for(long i = 2L; i <= select2; i++)
            m = m.multiply(BigInteger.valueOf(i));
        return n.divide(m);
    }
}
```

솔루션은 두 개의 BigInteger 변수인 n과 m을 1로 초기화합니다. 그런 다음 각각에 대한 루프를 사용하여 ball과 share의 계승을 계산합니다. 결과를 각각 n과 m에 저장합니다. 마지막으로 n을 m으로 나누고 결과를 반환합니다. 이 솔루션은 조합 수식의 결과가 매우 클 수 있고 long 또는 int 데이터 유형에 맞지 않을 수 있기 때문에 BigInteger를 사용합니다. BigInteger를 사용하면 매우 큰 정수를 저장하고 조작할 수 있습니다.