**AOC 2021 Questions**

**--- Day 18: Snailfish ---**

You descend into the ocean trench and encounter some [snailfish](https://en.wikipedia.org/wiki/Snailfish). They say they saw the sleigh keys! They'll even tell you which direction the keys went if you help one of the smaller snailfish with his math homework.

Snailfish numbers aren't like regular numbers. Instead, every snailfish number is a pair - an ordered list of two elements. Each element of the pair can be either a regular number or another pair.

Pairs are written as [x,y], where x and y are the elements within the pair. Here are some example snailfish numbers, one snailfish number per line:

[1,2]

[[1,2],3]

[9,[8,7]]

[[1,9],[8,5]]

[[[[1,2],[3,4]],[[5,6],[7,8]]],9]

[[[9,[3,8]],[[0,9],6]],[[[3,7],[4,9]],3]]

[[[[1,3],[5,3]],[[1,3],[8,7]]],[[[4,9],[6,9]],[[8,2],[7,3]]]]

This snailfish homework is about addition. To add two snailfish numbers, form a pair from the left and right parameters of the addition operator. For example, [1,2] + [[3,4],5] becomes [[1,2],[[3,4],5]].

There's only one problem: snailfish numbers must always be reduced, and the process of adding two snailfish numbers can result in snailfish numbers that need to be reduced.

To reduce a snailfish number, you must repeatedly do the first action in this list that applies to the snailfish number:

* If any pair is nested inside four pairs, the leftmost such pair explodes.
* If any regular number is 10 or greater, the leftmost such regular number splits.

Once no action in the above list applies, the snailfish number is reduced.

During reduction, at most one action applies, after which the process returns to the top of the list of actions. For example, if split produces a pair that meets the explode criteria, that pair explodes before other splits occur.

To explode a pair, the pair's left value is added to the first regular number to the left of the exploding pair (if any), and the pair's right value is added to the first regular number to the right of the exploding pair (if any). Exploding pairs will always consist of two regular numbers. Then, the entire exploding pair is replaced with the regular number 0.

Here are some examples of a single explode action:

* [[[[[9,8],1],2],3],4] becomes [[[[0,9],2],3],4] (the 9 has no regular number to its left, so it is not added to any regular number).
* [7,[6,[5,[4,[3,2]]]]] becomes [7,[6,[5,[7,0]]]] (the 2 has no regular number to its right, and so it is not added to any regular number).
* [[6,[5,[4,[3,2]]]],1] becomes [[6,[5,[7,0]]],3].
* [[3,[2,[1,[7,3]]]],[6,[5,[4,[3,2]]]]] becomes [[3,[2,[8,0]]],[9,[5,[4,[3,2]]]]] (the pair [3,2] is unaffected because the pair [7,3] is further to the left; [3,2] would explode on the next action).
* [[3,[2,[8,0]]],[9,[5,[4,[3,2]]]]] becomes [[3,[2,[8,0]]],[9,[5,[7,0]]]].

To split a regular number, replace it with a pair; the left element of the pair should be the regular number divided by two and rounded down, while the right element of the pair should be the regular number divided by two and rounded up. For example, 10 becomes [5,5], 11 becomes [5,6], 12 becomes [6,6], and so on.

Here is the process of finding the reduced result of [[[[4,3],4],4],[7,[[8,4],9]]] + [1,1]:

after addition: [[[[[4,3],4],4],[7,[[8,4],9]]],[1,1]]

after explode: [[[[0,7],4],[7,[[8,4],9]]],[1,1]]

after explode: [[[[0,7],4],[15,[0,13]]],[1,1]]

after split: [[[[0,7],4],[[7,8],[0,13]]],[1,1]]

after split: [[[[0,7],4],[[7,8],[0,[6,7]]]],[1,1]]

after explode: [[[[0,7],4],[[7,8],[6,0]]],[8,1]]

Once no reduce actions apply, the snailfish number that remains is the actual result of the addition operation: [[[[0,7],4],[[7,8],[6,0]]],[8,1]].

The homework assignment involves adding up a list of snailfish numbers (your puzzle input). The snailfish numbers are each listed on a separate line. Add the first snailfish number and the second, then add that result and the third, then add that result and the fourth, and so on until all numbers in the list have been used once.

For example, the final sum of this list is [[[[1,1],[2,2]],[3,3]],[4,4]]:

[1,1]

[2,2]

[3,3]

[4,4]

The final sum of this list is [[[[3,0],[5,3]],[4,4]],[5,5]]:

[1,1]

[2,2]

[3,3]

[4,4]

[5,5]

The final sum of this list is [[[[5,0],[7,4]],[5,5]],[6,6]]:

[1,1]

[2,2]

[3,3]

[4,4]

[5,5]

[6,6]

Here's a slightly larger example:

[[[0,[4,5]],[0,0]],[[[4,5],[2,6]],[9,5]]]

[7,[[[3,7],[4,3]],[[6,3],[8,8]]]]

[[2,[[0,8],[3,4]]],[[[6,7],1],[7,[1,6]]]]

[[[[2,4],7],[6,[0,5]]],[[[6,8],[2,8]],[[2,1],[4,5]]]]

[7,[5,[[3,8],[1,4]]]]

[[2,[2,2]],[8,[8,1]]]

[2,9]

[1,[[[9,3],9],[[9,0],[0,7]]]]

[[[5,[7,4]],7],1]

[[[[4,2],2],6],[8,7]]

The final sum [[[[8,7],[7,7]],[[8,6],[7,7]]],[[[0,7],[6,6]],[8,7]]] is found after adding up the above snailfish numbers:

[[[0,[4,5]],[0,0]],[[[4,5],[2,6]],[9,5]]]

+ [7,[[[3,7],[4,3]],[[6,3],[8,8]]]]

= [[[[4,0],[5,4]],[[7,7],[6,0]]],[[8,[7,7]],[[7,9],[5,0]]]]

[[[[4,0],[5,4]],[[7,7],[6,0]]],[[8,[7,7]],[[7,9],[5,0]]]]

+ [[2,[[0,8],[3,4]]],[[[6,7],1],[7,[1,6]]]]

= [[[[6,7],[6,7]],[[7,7],[0,7]]],[[[8,7],[7,7]],[[8,8],[8,0]]]]

[[[[6,7],[6,7]],[[7,7],[0,7]]],[[[8,7],[7,7]],[[8,8],[8,0]]]]

+ [[[[2,4],7],[6,[0,5]]],[[[6,8],[2,8]],[[2,1],[4,5]]]]

= [[[[7,0],[7,7]],[[7,7],[7,8]]],[[[7,7],[8,8]],[[7,7],[8,7]]]]

[[[[7,0],[7,7]],[[7,7],[7,8]]],[[[7,7],[8,8]],[[7,7],[8,7]]]]

+ [7,[5,[[3,8],[1,4]]]]

= [[[[7,7],[7,8]],[[9,5],[8,7]]],[[[6,8],[0,8]],[[9,9],[9,0]]]]

[[[[7,7],[7,8]],[[9,5],[8,7]]],[[[6,8],[0,8]],[[9,9],[9,0]]]]

+ [[2,[2,2]],[8,[8,1]]]

= [[[[6,6],[6,6]],[[6,0],[6,7]]],[[[7,7],[8,9]],[8,[8,1]]]]

[[[[6,6],[6,6]],[[6,0],[6,7]]],[[[7,7],[8,9]],[8,[8,1]]]]

+ [2,9]

= [[[[6,6],[7,7]],[[0,7],[7,7]]],[[[5,5],[5,6]],9]]

[[[[6,6],[7,7]],[[0,7],[7,7]]],[[[5,5],[5,6]],9]]

+ [1,[[[9,3],9],[[9,0],[0,7]]]]

= [[[[7,8],[6,7]],[[6,8],[0,8]]],[[[7,7],[5,0]],[[5,5],[5,6]]]]

[[[[7,8],[6,7]],[[6,8],[0,8]]],[[[7,7],[5,0]],[[5,5],[5,6]]]]

+ [[[5,[7,4]],7],1]

= [[[[7,7],[7,7]],[[8,7],[8,7]]],[[[7,0],[7,7]],9]]

[[[[7,7],[7,7]],[[8,7],[8,7]]],[[[7,0],[7,7]],9]]

+ [[[[4,2],2],6],[8,7]]

= [[[[8,7],[7,7]],[[8,6],[7,7]]],[[[0,7],[6,6]],[8,7]]]

To check whether it's the right answer, the snailfish teacher only checks the magnitude of the final sum. The magnitude of a pair is 3 times the magnitude of its left element plus 2 times the magnitude of its right element. The magnitude of a regular number is just that number.

For example, the magnitude of [9,1] is 3\*9 + 2\*1 = 29; the magnitude of [1,9] is 3\*1 + 2\*9 = 21. Magnitude calculations are recursive: the magnitude of [[9,1],[1,9]] is 3\*29 + 2\*21 = 129.

Here are a few more magnitude examples:

* [[1,2],[[3,4],5]] becomes 143.
* [[[[0,7],4],[[7,8],[6,0]]],[8,1]] becomes 1384.
* [[[[1,1],[2,2]],[3,3]],[4,4]] becomes 445.
* [[[[3,0],[5,3]],[4,4]],[5,5]] becomes 791.
* [[[[5,0],[7,4]],[5,5]],[6,6]] becomes 1137.
* [[[[8,7],[7,7]],[[8,6],[7,7]]],[[[0,7],[6,6]],[8,7]]] becomes 3488.

So, given this example homework assignment:

[[[0,[5,8]],[[1,7],[9,6]]],[[4,[1,2]],[[1,4],2]]]

[[[5,[2,8]],4],[5,[[9,9],0]]]

[6,[[[6,2],[5,6]],[[7,6],[4,7]]]]

[[[6,[0,7]],[0,9]],[4,[9,[9,0]]]]

[[[7,[6,4]],[3,[1,3]]],[[[5,5],1],9]]

[[6,[[7,3],[3,2]]],[[[3,8],[5,7]],4]]

[[[[5,4],[7,7]],8],[[8,3],8]]

[[9,3],[[9,9],[6,[4,9]]]]

[[2,[[7,7],7]],[[5,8],[[9,3],[0,2]]]]

[[[[5,2],5],[8,[3,7]]],[[5,[7,5]],[4,4]]]

The final sum is:

[[[[6,6],[7,6]],[[7,7],[7,0]]],[[[7,7],[7,7]],[[7,8],[9,9]]]]

The magnitude of this final sum is 4140.

Add up all of the snailfish numbers from the homework assignment in the order they appear. What is the magnitude of the final sum?

To begin, [get your puzzle input](https://adventofcode.com/2021/day/18/input).

양식의 맨 위

Answer:

양식의 맨 아래

You can also [Share] this puzzle.

**--- 18일차 : 달팽이고기 ---**

(ChatGPT 번역)

당신은 심해의 해구로 내려가며 달팽이 물고기들을 만납니다. 그들은 썰매 열쇠가 어디로 갔는지 보았다고 말합니다! 당신이 작은 달팽이 물고기 중 하나의 수학 숙제를 도와준다면, 그들은 열쇠가 간 방향을 알려줄 것입니다.

달팽이 물고기의 숫자는 일반적인 숫자와는 다릅니다. 대신에, 모든 달팽이 물고기 숫자는 쌍(pair)으로 이루어져 있습니다. 쌍은 두 개의 요소로 구성된 순서가 있는 목록입니다. 쌍의 각 요소는 정수일 수도 있고 또 다른 쌍일 수도 있습니다.

쌍은 [x,y]와 같은 형태로 쓰이며, 여기서 x와 y는 쌍 내의 요소를 나타냅니다. 아래는 예시로 든 달팽이 물고기 숫자들로, 각 줄에 하나의 달팽이 물고기 숫자가 나옵니다:

[1,2]

[[1,2],3]

[9,[8,7]]

[[1,9],[8,5]]

[[[[1,2],[3,4]],[[5,6],[7,8]]],9]

[[[9,[3,8]],[[0,9],6]],[[[3,7],[4,9]],3]]

[[[[1,3],[5,3]],[[1,3],[8,7]]],[[[4,9],[6,9]],[[8,2],[7,3]]]]

이 달팽이 물고기의 수학 숙제는 더하기에 관한 것입니다. 두 달팽이 물고기 숫자를 더하려면, 더하기 연산자의 왼쪽과 오른쪽 인수로부터 하나의 쌍을 형성합니다. 예를 들어, [1,2] + [[3,4],5]는 [[1,2],[[3,4],5]]가 됩니다.

단 하나의 문제가 있습니다: 달팽이 물고기 숫자는 항상 줄여야 하며, 두 달팽이 물고기 숫자를 더하는 과정에서 줄여야 할 달팽이 물고기 숫자가 생길 수 있습니다.

달팽이 물고기 숫자를 줄이려면, 해당 숫자에 적용되는 첫 번째 동작을 반복적으로 수행해야 합니다:

• 만약 네 쌍 안에 중첩된 쌍이 있다면, 가장 왼쪽에 있는 쌍이 폭발합니다.

• 만약 10 이상인 정수가 있다면, 가장 왼쪽에 있는 정수가 분할됩니다.

위의 동작 중 어느 것도 더 이상 적용되지 않을 때까지 숫자를 줄여야 합니다.

줄이는 동안 한 번에 하나의 동작만 적용되며, 동작이 완료되면 다시 처음부터 적용 가능한 동작을 찾습니다. 예를 들어, 분할이 폭발 조건을 만족하는 쌍을 생성하는 경우, 다른 분할이 발생하기 전에 해당 쌍이 먼저 폭발합니다.

쌍이 폭발하면, 쌍의 왼쪽 값은 폭발하는 쌍의 왼쪽에 있는 첫 번째 정수에 더해지고, 쌍의 오른쪽 값은 오른쪽에 있는 첫 번째 정수에 더해집니다. 폭발하는 쌍은 항상 두 개의 정수로 이루어져 있으며, 그 쌍 전체는 0으로 대체됩니다.

아래는 한 번의 폭발 동작의 예시입니다:

• [[[[[9,8],1],2],3],4]는 [[[[0,9],2],3],4]가 됩니다 (9는 왼쪽에 정수가 없으므로 더해지지 않습니다).

• [7,[6,[5,[4,[3,2]]]]]는 [7,[6,[5,[7,0]]]]가 됩니다 (2는 오른쪽에 정수가 없으므로 더해지지 않습니다).

• [[6,[5,[4,[3,2]]]],1]은 [[6,[5,[7,0]]],3]이 됩니다.

• [[3,[2,[1,[7,3]]]],[6,[5,[4,[3,2]]]]]는 [[3,[2,[8,0]]],[9,[5,[4,[3,2]]]]]가 됩니다 (쌍 [3,2]는 왼쪽에 있는 [7,3] 때문에 영향을 받지 않으며, [3,2]는 다음 동작에서 폭발하게 됩니다).

• [[3,[2,[8,0]]],[9,[5,[4,[3,2]]]]]는 [[3,[2,[8,0]]],[9,[5,[7,0]]]]이 됩니다.

정수를 분할하려면, 해당 정수를 쌍으로 대체합니다. 쌍의 왼쪽 요소는 해당 정수를 2로 나누어 버림한 값이고, 오른쪽 요소는 해당 정수를 2로 나눈 후 올림한 값입니다. 예를 들어, 10은 [5,5], 11은 [5,6], 12는 [6,6]과 같이 변환됩니다.

아래는 [[[[4,3],4],4],[7,[[8,4],9]]] + [1,1]의 축소 과정을 보여줍니다:

• 더한 후: [[[[[4,3],4],4],[7,[[8,4],9]]],[1,1]]

• 폭발 후: [[[[0,7],4],[7,[[8,4],9]]],[1,1]]

• 폭발 후: [[[[0,7],4],[15,[0,13]]],[1,1]]

• 분할 후: [[[[0,7],4],[[7,8],[0,13]]],[1,1]]

• 분할 후: [[[[0,7],4],[[7,8],[0,[6,7]]]],[1,1]]

• 폭발 후: [[[[0,7],4],[[7,8],[6,0]]],[8,1]]

더 이상 줄일 수 없을 때 남아있는 달팽이 물고기 숫자가 실제로 더한 결과입니다: [[[[0,7],4],[[7,8],[6,0]]],[8,1]].

숙제 과제는 달팽이 물고기 숫자 목록을 순차적으로 더하는 것입니다 (퍼즐 입력이 주어집니다). 첫 번째 달팽이 물고기 숫자와 두 번째 숫자를 더한 후, 그 결과와 세 번째 숫자를 더하고, 그 결과와 네 번째 숫자를 더하는 식으로 목록에 있는 모든 숫자를 한 번씩 사용할 때까지 계속 더합니다.

예를 들어, 아래 목록의 최종 합은 [[[[1,1],[2,2]],[3,3]],[4,4]]입니다:

[1,1]

[2,2]

[3,3]

[4,4]

다음 목록의 최종 합은 [[[[3,0],[5,3]],[4,4]],[5,5]]입니다:

[1,1]

[2,2]

[3,3]

[4,4]

[5,5]

다음 목록의 최종 합은 [[[[5,0],[7,4]],[5,5]],[6,6]]입니다:

[1,1]

[2,2]

[3,3]

[4,4]

[5,5]

[6,6]

좀 더 큰 예시입니다:

[[[0,[4,5]],[0,0]],[[[4,5],[2,6]],[9,5]]]

[7,[[[3,7],[4,3]],[[6,3],[8,8]]]]

[[2,[[0,8],[3,4]]],[[[6,7],1],[7,[1,6]]]]

[[[[2,4],7],[6,[0,5]]],[[[6,8],[2,8]],[[2,1],[4,5]]]]

[7,[5,[[3,8],[1,4]]]]

[[2,[2,2]],[8,[8,1]]]

[2,9]

[1,[[[9,3],9],[[9,0],[0,7]]]]

[[[5,[7,4]],7],1]

[[[[4,2],2],6],[8,7]]

위의 달팽이 물고기 숫자들을 모두 더한 최종 합은 [[[[8,7],[7,7]],[[8,6],[7,7]]],[[[0,7],[6,6]],[8,7]]]입니다.

최종 합계인 [[[[8,7],[7,7]],[[8,6],[7,7]]],[[[0,7],[6,6]],[8,7]]]는 위의 달팽이 물고기 숫자들을 더한 후에 나온 결과입니다:

[[[0,[4,5]],[0,0]],[[[4,5],[2,6]],[9,5]]]

+ [7,[[[3,7],[4,3]],[[6,3],[8,8]]]]

= [[[[4,0],[5,4]],[[7,7],[6,0]]],[[8,[7,7]],[[7,9],[5,0]]]]

[[[[4,0],[5,4]],[[7,7],[6,0]]],[[8,[7,7]],[[7,9],[5,0]]]]

+ [[2,[[0,8],[3,4]]],[[[6,7],1],[7,[1,6]]]]

= [[[[6,7],[6,7]],[[7,7],[0,7]]],[[[8,7],[7,7]],[[8,8],[8,0]]]]

[[[[6,7],[6,7]],[[7,7],[0,7]]],[[[8,7],[7,7]],[[8,8],[8,0]]]]

+ [[[[2,4],7],[6,[0,5]]],[[[6,8],[2,8]],[[2,1],[4,5]]]]

= [[[[7,0],[7,7]],[[7,7],[7,8]]],[[[7,7],[8,8]],[[7,7],[8,7]]]]

[[[[7,0],[7,7]],[[7,7],[7,8]]],[[[7,7],[8,8]],[[7,7],[8,7]]]]

+ [7,[5,[[3,8],[1,4]]]]

= [[[[7,7],[7,8]],[[9,5],[8,7]]],[[[6,8],[0,8]],[[9,9],[9,0]]]]

[[[[7,7],[7,8]],[[9,5],[8,7]]],[[[6,8],[0,8]],[[9,9],[9,0]]]]

+ [[2,[2,2]],[8,[8,1]]]

= [[[[6,6],[6,6]],[[6,0],[6,7]]],[[[7,7],[8,9]],[8,[8,1]]]]

[[[[6,6],[6,6]],[[6,0],[6,7]]],[[[7,7],[8,9]],[8,[8,1]]]]

+ [2,9]

= [[[[6,6],[7,7]],[[0,7],[7,7]]],[[[5,5],[5,6]],9]]

[[[[6,6],[7,7]],[[0,7],[7,7]]],[[[5,5],[5,6]],9]]

+ [1,[[[9,3],9],[[9,0],[0,7]]]]

= [[[[7,8],[6,7]],[[6,8],[0,8]]],[[[7,7],[5,0]],[[5,5],[5,6]]]]

[[[[7,8],[6,7]],[[6,8],[0,8]]],[[[7,7],[5,0]],[[5,5],[5,6]]]]

+ [[[5,[7,4]],7],1]

= [[[[7,7],[7,7]],[[8,7],[8,7]]],[[[7,0],[7,7]],9]]

[[[[7,7],[7,7]],[[8,7],[8,7]]],[[[7,0],[7,7]],9]]

+ [[[[4,2],2],6],[8,7]]

= [[[[8,7],[7,7]],[[8,6],[7,7]]],[[[0,7],[6,6]],[8,7]]]

올바른 답인지 확인하기 위해 달팽이 물고기 선생님은 최종 합계의 크기만 확인합니다. 쌍의 크기는 왼쪽 요소의 크기에 3을 곱하고 오른쪽 요소의 크기에 2를 곱한 값입니다. 정규 숫자의 크기는 그 숫자 그대로입니다.

예를 들어, [9,1]의 크기는 39 + 21 = 29이며, [1,9]의 크기는 31 + 29 = 21입니다. 크기 계산은 재귀적으로 이루어집니다. 예를 들어, [[9,1],[1,9]]의 크기는 329 + 221 = 129입니다.

다음은 몇 가지 크기 계산 예시입니다:

[[1,2],[[3,4],5]]는 143이 됩니다.

[[[[0,7],4],[[7,8],[6,0]]],[8,1]]은 1384가 됩니다.

[[[[1,1],[2,2]],[3,3]],[4,4]]은 445가 됩니다.

[[[[3,0],[5,3]],[4,4]],[5,5]]은 791이 됩니다.

[[[[5,0],[7,4]],[5,5]],[6,6]]은 1137이 됩니다.

[[[[8,7],[7,7]],[[8,6],[7,7]]],[[[0,7],[6,6]],[8,7]]]는 3488이 됩니다.

따라서, 주어진 숙제 예시의 크기는 다음과 같습니다:

[[[0,[5,8]],[[1,7],[9,6]]],[[4,[1,2]],[[1,4],2]]]

[[[5,[2,8]],4],[5,[[9,9],0]]]

[6,[[[6,2],[5,6]],[[7,6],[4,7]]]]

[[[6,[0,7]],[0,9]],[4,[9,[9,0]]]]

[[[7,[6,4]],[3,[1,3]]],[[[5,5],1],9]]

[[6,[[7,3],[3,2]]],[[[3,8],[5,7]],4]]

[[[[5,4],[7,7]],8],[[8,3],8]]

[[9,3],[[9,9],[6,[4,9]]]]

[[2,[[7,7],7]],[[5,8],[[9,3],[0,2]]]]

[[[[5,2],5],[8,[3,7]]],[[5,[7,5]],[4,4]]]

최종 합계는:

[[[[6,6],[7,6]],[[7,7],[7,0]]],[[[7,7],[7,7]],[[7,8],[9,9]]]]

이 최종 합계의 크기는 4140입니다.

주어진 숙제에서 달팽이 물고기 숫자들을 순서대로 모두 더하세요. 최종 합계의 크기는 얼마입니까?

시작하려면 퍼즐 입력을 받으세요.

--- Day 17: Trick Shot ---

You finally decode the Elves' message. HI, the message says. You continue searching for the sleigh keys.

Ahead of you is what appears to be a large [ocean trench](https://en.wikipedia.org/wiki/Oceanic_trench). Could the keys have fallen into it? You'd better send a probe to investigate.

The probe launcher on your submarine can fire the probe with any [integer](https://en.wikipedia.org/wiki/Integer) velocity in the x (forward) and y (upward, or downward if negative) directions. For example, an initial x,y velocity like 0,10 would fire the probe straight up, while an initial velocity like 10,-1 would fire the probe forward at a slight downward angle.

The probe's x,y position starts at 0,0. Then, it will follow some trajectory by moving in steps. On each step, these changes occur in the following order:

* The probe's x position increases by its x velocity.
* The probe's y position increases by its y velocity.
* Due to drag, the probe's x velocity changes by 1 toward the value 0; that is, it decreases by 1 if it is greater than 0, increases by 1 if it is less than 0, or does not change if it is already 0.
* Due to gravity, the probe's y velocity decreases by 1.

For the probe to successfully make it into the trench, the probe must be on some trajectory that causes it to be within a target area after any step. The submarine computer has already calculated this target area (your puzzle input). For example:

target area: x=20..30, y=-10..-5

This target area means that you need to find initial x,y velocity values such that after any step, the probe's x position is at least 20 and at most 30, and the probe's y position is at least -10 and at most -5.

Given this target area, one initial velocity that causes the probe to be within the target area after any step is 7,2:

.............#....#............

.......#..............#........

...............................

S........................#.....

...............................

...............................

...........................#...

...............................

....................TTTTTTTTTTT

....................TTTTTTTTTTT

....................TTTTTTTT#TT

....................TTTTTTTTTTT

....................TTTTTTTTTTT

....................TTTTTTTTTTT

In this diagram, S is the probe's initial position, 0,0. The x coordinate increases to the right, and the y coordinate increases upward. In the bottom right, positions that are within the target area are shown as T. After each step (until the target area is reached), the position of the probe is marked with #. (The bottom-right # is both a position the probe reaches and a position in the target area.)

Another initial velocity that causes the probe to be within the target area after any step is 6,3:

...............#..#............

...........#........#..........

...............................

......#..............#.........

...............................

...............................

S....................#.........

...............................

...............................

...............................

.....................#.........

....................TTTTTTTTTTT

....................TTTTTTTTTTT

....................TTTTTTTTTTT

....................TTTTTTTTTTT

....................T#TTTTTTTTT

....................TTTTTTTTTTT

Another one is 9,0:

S........#.....................

.................#.............

...............................

........................#......

...............................

....................TTTTTTTTTTT

....................TTTTTTTTTT#

....................TTTTTTTTTTT

....................TTTTTTTTTTT

....................TTTTTTTTTTT

....................TTTTTTTTTTT

One initial velocity that doesn't cause the probe to be within the target area after any step is 17,-4:

S..............................................................

...............................................................

...............................................................

...............................................................

.................#.............................................

....................TTTTTTTTTTT................................

....................TTTTTTTTTTT................................

....................TTTTTTTTTTT................................

....................TTTTTTTTTTT................................

....................TTTTTTTTTTT..#.............................

....................TTTTTTTTTTT................................

...............................................................

...............................................................

...............................................................

...............................................................

................................................#..............

...............................................................

...............................................................

...............................................................

...............................................................

...............................................................

...............................................................

..............................................................#

The probe appears to pass through the target area, but is never within it after any step. Instead, it continues down and to the right - only the first few steps are shown.

If you're going to fire a highly scientific probe out of a super cool probe launcher, you might as well do it with style. How high can you make the probe go while still reaching the target area?

In the above example, using an initial velocity of 6,9 is the best you can do, causing the probe to reach a maximum y position of 45. (Any higher initial y velocity causes the probe to overshoot the target area entirely.)

Find the initial velocity that causes the probe to reach the highest y position and still eventually be within the target area after any step. What is the highest y position it reaches on this trajectory?

To begin, [get your puzzle input](https://adventofcode.com/2021/day/17/input).

양식의 맨 위

Answer:

양식의 맨 아래

You can also [Share] this puzzle.

**--- Day 17: Trick Shot ---**

드디어 엘프들의 메시지를 해독했습니다. 메시지에는 "HI"라고 적혀 있습니다. 이제 썰매 열쇠를 계속 찾아야 합니다.

앞에 큰 바다 해구가 보입니다. 열쇠가 그 안으로 떨어졌을까요? 탐사 장비를 보내서 조사 해 봐야 겠습니다.

잠수함의 탐사 장비 발사기는 x축(앞 방향)과 y축(위쪽 또는 아래쪽 방향, y값이 음수일 때)으로 초기 속도를 임의의 정수 값으로 설정해 발사할 수 있습니다. 예를 들어, 초기 속도가 0,10인 경우 탐사 장비는 직선으로 위로 발사되고, 초기 속도가 10,-1인 경우 약간 아래쪽으로 기울어진 각도로 앞으로 발사됩니다.

탐사 장비의 x,y 위치는 (0,0)에서 시작합니다. 그 후, 탐사 장비는 몇 단계에 걸쳐 다음과 같은 순서로 움직입니다:

1. 탐사 장비의 x 위치는 현재 x 속도만큼 증가합니다.

2. 탐사 장비의 y 위치는 현재 y 속도만큼 증가합니다.

3. 항력에 의해 탐사 장비의 x 속도는 0에 가까워지도록 1만큼 변합니다. 즉, x 속도가 0보다 크면 1만큼 감소하고, 0보다 작으면 1만큼 증가하며, 이미 0인 경우 변화하지 않습니다.

4. 중력에 의해 탐사 장비의 y 속도는 1만큼 감소합니다.

탐사 장비가 해구에 성공적으로 도달하기 위해서는, 탐사 장비가 어떤 단계에서든 목표 영역 내에 도달해야 합니다. 잠수함 컴퓨터는 이미 이 목표 영역(퍼즐 입력값)을 계산해 두었습니다. 예를 들어:

```

목표 영역: x=20..30, y=-10..-5

```

이 목표 영역은 탐사 장비의 위치가 x 좌표에서 최소 20 이상 30 이하, y 좌표에서 최소 -10 이상 -5 이하가 되도록 하는 초기 x, y 속도를 찾아야 한다는 의미입니다.

예를 들어, 초기 속도 7,2는 탐사 장비가 어느 단계에서 목표 영역에 도달할 수 있게 합니다:

.............#....#............

.......#..............#........

...............................

S........................#.....

...............................

...............................

...........................#...

...............................

....................TTTTTTTTTTT

....................TTTTTTTTTTT

....................TTTTTTTT#TT

....................TTTTTTTTTTT

....................TTTTTTTTTTT

....................TTTTTTTTTTT

이 다이어그램에서 S는 탐사 장비의 초기 위치 (0,0)를 나타냅니다. x 좌표는 오른쪽으로 증가하고, y 좌표는 위로 증가합니다. 오른쪽 아래에는 목표 영역 내에 있는 위치가 T로 표시되어 있습니다. 각 단계가 끝난 후(목표 영역에 도달할 때까지) 탐사 장비의 위치는 #으로 표시됩니다. (오른쪽 아래의 #는 탐사 장비가 도달한 위치이자 목표 영역 내에 있는 위치입니다.)

초기 속도 6,3도 목표 영역에 도달할 수 있습니다:

...............#..#............

...........#........#..........

...............................

......#..............#.........

...............................

...............................

S....................#.........

...............................

...............................

...............................

.....................#.........

....................TTTTTTTTTTT

....................TTTTTTTTTTT

....................TTTTTTTTTTT

....................TTTTTTTTTTT

....................T#TTTTTTTTT

....................TTTTTTTTTTT

초기 속도 9,0도 마찬가지입니다:

```

S........#.....................

.................#.............

...............................

........................#......

...............................

....................TTTTTTTTTTT

....................TTTTTTTTTT#

....................TTTTTTTTTTT

....................TTTTTTTTTTT

....................TTTTTTTTTTT

....................TTTTTTTTTTT

목표 영역을 통과하지만 어느 단계에서도 그 영역 내에 도달하지 못하는 초기 속도는 17,-4입니다:

S..............................................................

...............................................................

...............................................................

...............................................................

.................#.............................................

....................TTTTTTTTTTT................................

....................TTTTTTTTTTT................................

....................TTTTTTTTTTT................................

....................TTTTTTTTTTT................................

....................TTTTTTTTTTT..#.............................

....................TTTTTTTTTTT................................

...............................................................

...............................................................

...............................................................

...............................................................

................................................#..............

...............................................................

...............................................................

...............................................................

...............................................................

...............................................................

...............................................................

..............................................................#

탐사 장비는 목표 영역을 통과한 것처럼 보이지만, 어느 단계에서도 그 안에 도달하지 못합니다. 대신 계속해서 오른쪽 아래로 내려갑니다. 위에는 몇 단계만 보여주었습니다.

멋진 탐사 장비 발사기로 과학적 탐사를 할 때, 스타일리시하게 해야겠죠. 목표 영역에 도달하면서 탐사 장비가 도달할 수 있는 최고 높이는 얼마일까요?

위의 예에서, 초기 속도 6,9를 사용하면 탐사 장비는 최대 y 위치 45에 도달할 수 있습니다. (그 이상으로 초기 y 속도를 높이면 탐사 장비가 목표 영역을 완전히 벗어나게 됩니다.)

목표 영역에 도달하면서 탐사 장비가 도달할 수 있는 최고 높이는 얼마일까요?

--- Day 16: 패킷 디코더 ---

동굴을 떠나 넓은 바다에 도착하자, 엘프들이 배에서 보내온 전송 메시지를 받았습니다.

이 메시지는 부력 교환 전송 시스템(BITS)을 사용하여 전송된 것으로, 숫자 표현식을 이진수 시퀀스로 압축하는 방법입니다. 잠수함의 컴퓨터는 이 메시지를 16진수(퍼즐 입력값)로 저장했습니다.

메시지를 디코딩하는 첫 번째 단계는 16진수 표현을 이진수로 변환하는 것입니다. 16진수의 각 문자는 4비트의 이진 데이터에 해당합니다:

```

0 = 0000

1 = 0001

2 = 0010

3 = 0011

4 = 0100

5 = 0101

6 = 0110

7 = 0111

8 = 1000

9 = 1001

A = 1010

B = 1011

C = 1100

D = 1101

E = 1110

F = 1111

```

BITS 전송은 가장 바깥쪽에 단일 패킷을 포함하고 있으며, 이 패킷 안에는 여러 개의 다른 패킷이 포함될 수 있습니다. 이 패킷의 16진수 표현에는 몇 개의 추가적인 0비트가 끝에 포함될 수 있으며, 이는 전송의 일부가 아니므로 무시해야 합니다.

모든 패킷은 표준 헤더로 시작합니다: 처음 세 비트는 패킷 버전을 인코딩하고, 다음 세 비트는 패킷 유형 ID를 인코딩합니다. 이 두 값은 숫자이며, 패킷에 인코딩된 모든 숫자는 가장 중요한 비트가 먼저 오는 방식으로 이진수로 표현됩니다. 예를 들어, 100으로 인코딩된 버전은 숫자 4를 나타냅니다.

패킷 유형 ID가 4인 패킷은 리터럴 값을 나타냅니다. 리터럴 값 패킷은 단일 이진수를 인코딩합니다. 이를 위해 이진수는 4비트의 배수가 될 때까지 앞에 0을 추가하여 패딩된 후 4비트씩 그룹으로 나뉩니다. 각 그룹은 1비트로 시작되며, 마지막 그룹은 0비트로 시작됩니다. 이러한 5비트 그룹은 패킷 헤더 바로 뒤에 나옵니다. 예를 들어, 16진수 문자열 `D2FE28`은 다음과 같이 변환됩니다:

```

110100101111111000101000

VVVTTTAAAAABBBBBCCCCC

```

각 비트 아래에 그 목적이 표시되어 있습니다:

- V로 표시된 처음 세 비트(110)는 패킷 버전 6을 나타냅니다.

- T로 표시된 다음 세 비트(100)는 패킷 유형 ID 4를 나타내며, 이는 패킷이 리터럴 값임을 의미합니다.

- A로 표시된 다섯 비트(10111)는 1로 시작되며(마지막 그룹이 아님) 숫자의 처음 네 비트를 포함하고 있습니다: 0111.

- B로 표시된 다섯 비트(11110)는 1로 시작되며(마지막 그룹이 아님) 숫자의 네 비트를 더 포함합니다: 1110.

- C로 표시된 다섯 비트(00101)는 0으로 시작되며(마지막 그룹, 패킷의 끝) 숫자의 마지막 네 비트를 포함하고 있습니다: 0101.

- 마지막에 있는 세 개의 표시되지 않은 0비트는 16진수 표현 때문에 추가된 것으로 무시해야 합니다.

따라서 이 패킷은 011111100101라는 이진수 표현을 가진 리터럴 값을 나타내며, 이는 10진수로 2021입니다.

패킷 유형 ID가 4가 아닌 다른 모든 패킷은 포함된 하나 이상의 하위 패킷에서 계산을 수행하는 연산자를 나타냅니다. 지금은 특정 연산이 중요하지 않으니, 하위 패킷의 계층 구조를 파싱하는 데 집중하세요.

연산자 패킷은 하나 이상의 하위 패킷을 포함합니다. 이 하위 패킷을 나타내는 이진 데이터가 무엇인지 나타내기 위해, 연산자 패킷은 패킷 헤더 바로 다음에 나오는 비트로 하위 패킷의 길이 유형 ID를 표시합니다:

- 길이 유형 ID가 0인 경우, 다음 15비트는 이 패킷에 포함된 하위 패킷의 전체 길이(비트 수)를 나타내는 숫자입니다.

- 길이 유형 ID가 1인 경우, 다음 11비트는 이 패킷에 바로 포함된 하위 패킷의 개수를 나타내는 숫자입니다.

마지막으로, 길이 유형 ID 비트와 15비트 또는 11비트 필드 뒤에 하위 패킷들이 나타납니다.

예를 들어, 길이 유형 ID 0을 사용하여 두 개의 하위 패킷을 포함하는 연산자 패킷(16진수 문자열 `38006F45291200`)은 다음과 같습니다:

```

00111000000000000110111101000101001010010001001000000000

VVVTTTILLLLLLLLLLLLLLLAAAAAAAAAAABBBBBBBBBBBBBBBB

```

- V로 표시된 처음 세 비트(001)는 패킷 버전 1을 나타냅니다.

- T로 표시된 다음 세 비트(110)는 패킷 유형 ID 6을 나타내며, 이는 이 패킷이 연산자임을 의미합니다.

- I로 표시된 비트(0)는 길이 유형 ID로, 이 패킷에 포함된 하위 패킷의 길이가 15비트 숫자로 표시된다는 것을 나타냅니다.

- L로 표시된 15비트(000000000011011)는 하위 패킷의 길이(비트 수)를 나타내며, 이는 27입니다.

- A로 표시된 11비트는 첫 번째 하위 패킷을 포함하며, 이는 10을 나타내는 리터럴 값입니다.

- B로 표시된 16비트는 두 번째 하위 패킷을 포함하며, 이는 20을 나타내는 리터럴 값입니다.

11비트와 16비트의 하위 패킷 데이터를 읽은 후, L에 표시된 총 길이(27)가 도달하면 이 패킷의 파싱이 종료됩니다.

또 다른 예로, 길이 유형 ID 1을 사용하여 세 개의 하위 패킷을 포함하는 연산자 패킷(16진수 문자열 `EE00D40C823060`)은 다음과 같습니다:

```

11101110000000001101010000001100100000100011000001100000

VVVTTTILLLLLLLLLLLAAAAAAAAAAABBBBBBBBBBBCCCCCCCCCCC

```

- V로 표시된 처음 세 비트(111)는 패킷 버전 7을 나타냅니다.

- T로 표시된 다음 세 비트(011)는 패킷 유형 ID 3을 나타내며, 이는 이 패킷이 연산자임을 의미합니다.

- I로 표시된 비트(1)는 길이 유형 ID로, 이 패킷에 포함된 하위 패킷의 개수가 11비트 숫자로 표시된다는 것을 나타냅니다.

- L로 표시된 11비트(00000000011)는 하위 패킷의 개수를 나타내며, 이는 3입니다.

- A로 표시된 11비트는 첫 번째 하위 패킷을 포함하며, 이는 1을 나타내는 리터럴 값입니다.

- B로 표시된 11비트는 두 번째 하위 패킷을 포함하며, 이는 2를 나타내는 리터럴 값입니다.

- C로 표시된 11비트는 세 번째 하위 패킷을 포함하며, 이는 3을 나타내는 리터럴 값입니다.

세 개의 하위 패킷을 모두 읽은 후, L에 표시된 하위 패킷의 수(3)가 도달하면 이 패킷의 파싱이 종료됩니다.

이제, 전송 메시지 전체의 패킷 계층 구조를 파싱하고 모든 패킷의 버전 번호를 합산하세요.

아래는 16진수로 인코딩된 전송 메시지의 몇 가지 추가 예시입니다:

- `8A004A801A8002F478`은 연산자 패킷(버전 4)으로, 연산자 패킷(버전 1)을 포함하고 있으며, 그 안에는 연산자 패킷(버전 5)이 있고, 그 안에는 리터럴 값(버전 6)이 포함되어 있습니다. 이 패킷의 버전 합계는 16입니다.

- `620080001611562C8802118E34`은 연산자 패킷(버전 3)으로, 두 개의 하위 패킷을 포함하고 있으며, 각 하위 패킷은 두 개의 리터럴 값을 포함하는 연산자 패킷입니다. 이 패킷의 버전 합계는 12입니다.

- `C0015000016115A2E0802F182340`은 이전 예시와 동일한 구조를 가지고 있지만, 가장 바깥쪽 패킷이 다른 길이 유형 ID를 사용합니다. 이 패킷의 버전 합계는 23입니다.

- `A0016C880162017C3686B18A3D4780`은 연산자 패킷으로, 연산자 패킷을 포함하며, 그 안에는 다섯 개의 리터럴 값을 포함하는 연산자 패킷이 있습니다. 이 패킷의 버전 합계는 31입니다.

당신의 16진수로 인코딩된 BITS 전송 메시지를 디코딩하여 모든 패킷의 버전 번호를 더한 값을 구하세요.

--- Day 10: Syntax Scoring ---

You ask the submarine to determine the best route out of the deep-sea cave, but it only replies:

Syntax error in navigation subsystems on line: all of them

All of them?! The damage is worse than you thought. You bring up a copy of the navigation subsystem (your puzzle input).

The navigation subsystem syntax is made of several lines containing chunks. There are one or more chunks on each line, and chunks contain zero or more other chunks. Adjacent chunks are not separated by any delimiter; if one chunk stops, the next chunk (if any) can immediately start. Every chunk must open and close with one of four legal pairs of matching characters:

* If a chunk opens with (, it must close with ).
* If a chunk opens with [, it must close with ].
* If a chunk opens with {, it must close with }.
* If a chunk opens with <, it must close with >.

So, () is a legal chunk that contains no other chunks, as is []. More complex but valid chunks include ([]), {()()()}, <([{}])>, [<>({}){}[([])<>]], and even (((((((((()))))))))).

Some lines are incomplete, but others are corrupted. Find and disregard the corrupted lines first.

A corrupted line is one where a chunk closes with the wrong character – that is, where the characters it opens and closes with do not form one of the four legal pairs listed above.

Examples of corrupted chunks include, (), {()()()>, (((()))), and <({}){()}[{}]. Such a chunk can appear anywhere within a line, and its presence causes the whole line to be considered corrupted.

For example, consider the following navigation subsystem:

[({(<(())[]>[[{[]{<()<>>

[(()[<>])]({[<{<<[]>>(

{([(<{}[<>[]}>{[]{[(<()>

(((({<>}<{<{<>}{[]{[]{}

[[<[([]))<([[{}[[()]]]

[{[{({}]{}}([{[{{{}}([]

{<[[]]>}<{[{[{[]{()[[[]

[<(<(<(<{}))><([]([]()

<{([([[(<>()){}]>(<<{{

<{([{{}}[<[[[<>{}]]]>[]]

Some of the lines aren’t corrupted, just incomplete; you can ignore these lines for now. The remaining five lines are corrupted:

* {([(<{}[<>[]}>{[]{[(<()> - Expected ], but found } instead.
* [[<[([]))<([[{}[[()]]] - Expected ], but found ) instead.
* [{[{({}]{}}([{[{{{}}([] - Expected ), but found ] instead.
* [<(<(<(<{}))><([]([]() - Expected >, but found ) instead.
* <{([([[(<>()){}]>(<<{{ - Expected ], but found > instead.

Stop at the first incorrect closing character on each corrupted line.

Did you know that syntax checkers actually have contests to see who can get the high score for syntax errors in a file? It’s true! To calculate the syntax error score for a line, take the first illegal character on the line and look it up in the following table:

* ): 3 points.
* ]: 57 points.
* }: 1197 points.
* >: 25137 points.

CHAT GPT-3

Q: RUST 언어에서 blanket implementation이 무슨 뜻이고 어떻게 만드는지 또는 어떻게 생성이 되는지, 그리고 어떻게 사용하는지 설명 부탁

A:

Rust 언어에서의 “blanket implementation”은 특정 트레이트(traits)에 대해 모든 타입에 대해 자동으로 구현되는 것을 의미 합니다. 이는 Rust의 강력한 트레이트 시스템의 한 측면으로, 코드 재사용성과 일반성을 높이는 중요한 개념 입니다.

Blanket Implementation 생성

Blanket Implementation은 impl 키워드를 사용하여 트레이트에 대한 모든 타입에 대해 한 번에 구현 됩니다. 예를 들어, Display트레이트를 예로 들어 보겠습니다. 이 트레이트는 타입을 문자열로 변환하여 출력 할 수 있는 기능을 제공합니다.

// Blanket implementation for Display trait for all types that implement Debug

impl<T: std::fmt::Debug> std::fmt::Display for T {

fn fmt(&self, f: &mut std::fmt::Formatter<'\_>) -> std::fmt::Result {

write!(f, "{:?}", self)

}

}

위의 예제에서는 Display트레이트를 Debug트레이트를 구현한 모든 타입에 대해 자동으로 구현하도록 만들었습니다. 이제 Display트레이트의 fmt메서드를 호출하여 모든 Debug를 구현한 타입을 문자열로 변환 할 수 있습니다.

Blanket Implementation 사용

Blanket Implementation은 Rust코드에서 자동으로 적용되므로 별도의 설정이나 호출 없이 사용 할 수 있습니다. 예를 들어, 다음과 같이 Display트레이트를 사용하여 출력 할 수 있습니다.

fn main() {

let number = 42;

println!("Number: {}", number); // Prints: Number: 42

let message = "Hello, Rust!";

println!("{}", message); // Prints: Hello, Rust!

}

위의 예제에서는 number와 message변수는 각각i32와 &str타입 입니다. 이들은 Display트레이트를 구현한 타입이므로, prtinln! 매크로를 통해 쉽게 문자열로 변환되어 출력 됩니다.

주의사항

Blanket Implementation은 Rust의 강력한 기능이지만 너무 범용적으로 사용하면 의도하지 않은 결과를 초래 할 수 있습니다. 따라서 필요한 경우에만 적절히 사용하는 것이 좋습니다. 또한, 특정 타입에 대해 명시적으로 구현된 트레이트가 있는 경우에는 blanket implementation이 이를 오버라이드 할 수 있으므로 주의가 필요합니다.

요약하면, Rust의 blanket implementation은 트레이트를 자동으로 모든 타입에 대해 구현하는 강력한 기능으로 코드 재사용성과 유연성을 높이는 데 유용 합니다.

Numpy np.squeeze() 함수 : 크기가 1인 axis제거

이번시간에는 Python Numpy 배열에서 크기가 1인 추가 axis를 제거하는 np.squeeze()함수의 사용법과 의미에 대해서 간단히 살펴보자.

먼저, 다음과 같은 2차원 배열처럼 보이는 3차원 배열이 있다고 가정 해보자.

Import numpy as np

a = np.array([[[1, 2, 3],

[4, 5, 6],

[7, 8, 9]]])

a.shape # == (1, 3, 3)

겉보기에는 3 x 3크기의 2차원 배열처럼 보이나.

괄호가 가장 바깥쪽에 1개가 추가로 있어, 실제 shape은 (3, 3)이 아닌 (1, 3, 3)이다.

[1, 2, 3, 4]와 [[1, 2, 3, 4]]관계로 생각하면 이해가 쉽다.

np.squeeze() 함수의 기능은 [[1,2,3,4]]를 [1,2,3,4]의 형태로 바꿔주는 것으로 이해 하면 된다.

실제 예를 보자.

np.squeeze()함수의 기본 사용법

np.squeeze() 함수의 사용법은 np.squeeze(배열) 또는 배열.squeeze() 형태로 지정해주면 간단히 완료 된다.

a.squeeze() 또는 np.squeeze(a) 로 할 수 있다.

'''array([[1, 2, 3],

[4, 5, 6],

[7, 8, 9]])'''

a.squeeze().shape() # (3, 3)

이번에는 더 복잡한 경우로, 2차원 배열처럼 보이는 4차원 배열의 예이다.

B = np.array([[[[1, 2, 3,]],

[[4, 5, 6]],

[[7, 8, 9]]]])

b.shape # (1, 3, 1, 3)

b.squeeze()

‘’’array([[1,2,3],

[4,5,6],

[7,8,9]]), shape=(3,3)’’’

a배열보다 안쪽 축에도 괄호가 1개씩 더 있는 상태로,

(1,3,1,3)형태의 shape을 가지고 있었다 (axis=0,2의 자리크기가 1)

이 경우에도 squeeze()함수 1번으로 크기가 1인 axis자리를 모두 제거하여 2차원 배열로 변환이 성공적으로 이루어졌다.

np.squeeze()함수 axis설정

b배열에서 크기가 1인 모두 추가 axis자리가 전부 제거 되었는데,

일부 axis만 삭제하고 싶은 경우 axis인자를 설정 해 주면 된다.

np.squeeze(배열,axis) 또는 배열.squeeze(axis) 형태로 지정이 가능하다.

b배열에서 일부 axis만 제거하는 예시를 살펴 보자.

b.squeeze(axis=0) # == np.squeeze(b, axis=0)

‘’’array([[[1,2,3]],

[[4,5,6]],

[[7,8,9]]]), shape = (3,1,3)’’’

b.squeeze(axis=2)

‘’’arrya([[[1,2,3],

[4,5,6],

[7,8,9]]]), shape = (1,3,3)’’’