Day 7

Part 1

The Historians take you to a familiar rope bridge over a river in the middle of a jungle. The Chief isn't on this side of the bridge, though; maybe he's on the other side?

When you go to cross the bridge, you notice a group of engineers trying to repair it. (Apparently, it breaks pretty frequently.) You won't be able to cross until it's fixed.

You ask how long it'll take; the engineers tell you that it only needs final calibrations, but some young elephants were playing nearby and stole all the operators from their calibration equations! They could finish the calibrations if only someone could determine which test values could possibly be produced by placing any combination of operators into their calibration equations (your puzzle input).

For example:

```
190: 10 19
3267: 81 40 27
83: 17 5
156: 15 6
7290: 6 8 6 15
161011: 16 10 13
192: 17 8 14
21037: 9 7 18 13
292: 11 6 16 20
```

Each line represents a single equation. The test value appears before the colon on each line; it is your job to determine whether the remaining numbers can be combined with operators to produce the test value.

Operators are always evaluated left-to-right, not according to precedence rules. Furthermore, numbers in the equations cannot be rearranged. Glancing into the jungle, you can see elephants holding two different types of operators: add (`+`) and multiply (`*`).

Only three of the above equations can be made true by inserting operators:

- `190: 10 19` has only one position that accepts an operator: between 10 and 19. Choosing + would give 29, but choosing * would give the test value (10 * 19 = 190).
- `3267: 81 40 27` has two positions for operators. Of the four possible configurations of the operators, two cause the right side to match the test value: 81 + 40 * 27 and 81 * 40 + 27 both equal 3267 (when evaluated left-to-right)!
- `292: 11 6 16 20` can be solved in exactly one way: 11 + 6 * 16 + 20.

The engineers just need the total calibration result, which is the sum of the test values from just the equations that could possibly be true. In the above example, the sum of the test values for the three equations listed above is `3749` .

Determine which equations could possibly be true. What is their total calibration result?

In [1]:
```
from operator import add, mul
from pathlib import Path
```

In [2]:

```
test_path = Path("data/day07_test.txt")
data_path = Path("data/day07_data.txt")

def load_data(fpath: Path) -> list[tuple[int, list[int]]]:
    """Returns a list of (target, values) tuples"""
    output = []

    for line in fpath.open().readlines():
        data = line.strip().split(": ")
        target = int(data[0])
        vals = [int(_) for _ in data[1].split()]
        output.append((vals, target))

    return output


def test(vals: list[int], target:int, curval: int|None=None, ops: list[str]|None=None
    """Returns True if the operator/values combination makes the target.

    Depth-first search exploring all combinations of operators
    until the target value is reached; returns False if the
    target is not reached.
    """
    # Top level
    if curval is None:
        curval = vals.pop(0)
        ops = []

    # End of all elements
    if len(vals) == 0:
        return False, ops

    # Get value to work with
    val = vals.pop(0)

    # Does addition hit the target?
    ops.append("+")
    if add(curval, val) == target and len(vals) == 0:
        return True, ops  # We have a solution

    # Addition didn't work, descend the tree
    retval, retops = test(vals[:], target, add(curval, val), ops[:])
    if retval:  # Descent found a solution, return it
        return retval, retops
    ops.pop()  # Discard addition

    # Descending the tree didn't work, try multiplication
    ops.append("*")
    if mul(curval, val) == target and len(vals) == 0:
        return True, ops  # We have a solution

    # Multiplication didn't work, descend the tree
    retval, retops = test(vals[:], target, mul(curval, val), ops[:])
    if retval:  # Descent found a solution, return it
        return retval, retops
    ops.pop()  # Discard multiplication
    return retval, ops


def test_data(data: list[tuple[int, list[int]]]) -> list[tuple[tuple[bool, list[str]]
    """Returns list of test results and corresponding targets"""
    results = []

    for vals, target in data:
```

```
        results.append((test(vals[:], target), target))

    return results


def get_total(data: list[tuple[int, list[int]]]) -> int:
    """Returns sum of targets for equations passing the test"""
    result = test_data(data)
    return sum([int(target) for (retval, target) in result if retval[0]])


data = load_data(test_path)
get_total(data)
```

Out[2]: 3749

In [3]:
```
data = load_data(data_path)
get_total(data)
```

Out[3]: 20281182715321

Part 2

The engineers seem concerned; the total calibration result you gave them is nowhere close to being within safety tolerances. Just then, you spot your mistake: some well-hidden elephants are holding a third type of operator.

The concatenation operator (`||`) combines the digits from its left and right inputs into a single number. For example, `12 || 345` would become `12345`. All operators are still evaluated left-to-right.

Now, apart from the three equations that could be made true using only addition and multiplication, the above example has three more equations that can be made true by inserting operators:

- `156: 15 6` can be made true through a single concatenation: 15 || 6 = 156.
- `7290: 6 8 6 15` can be made true using 6 * 8 || 6 * 15.
- `192: 17 8 14` can be made true using 17 || 8 + 14.

Adding up all six test values (the three that could be made before using only `+` and `*` plus the new three that can now be made by also using `||`) produces the new total calibration result of `11387`.

Using your new knowledge of elephant hiding spots, determine which equations could possibly be true. What is their total calibration result?

In [4]:
```
def concat(left: int, right: int) -> int:
    """Returns the result of concatenating two integers as strings."""
    return int(str(left) + str(right))


def test_concat(vals, target, curval=None, ops=None):
    """Returns True if the operator/values combination makes the target.

    Depth-first search exploring all combinations of operators
    until the target value is reached; returns False if the
    target is not reached.

    Adds the concatenation operator, with respect to the original solution
    """
    # Top level
```

```
    if curval is None:
        curval = vals.pop(0)
        ops = []

    # End of all elements
    if len(vals) == 0:
        return False, ops

    # Get value to work with
    val = vals.pop(0)

    # Does addition hit the target?
    ops.append("+")
    if add(curval, val) == target and len(vals) == 0:
        return True, ops  # We have a solution

    # Addition didn't work, descend the tree
    retval, retops = test_concat(vals[:], target, add(curval, val), ops[:])
    if retval:  # Descent found a solution, return it
        return retval, retops
    ops.pop()  # Discard addition

    # Descending the tree didn't work, try multiplication
    ops.append("*")
    if mul(curval, val) == target and len(vals) == 0:
        return True, ops  # We have a solution

    # Multiplication didn't work, descend the tree
    retval, retops = test_concat(vals[:], target, mul(curval, val), ops[:])
    if retval:  # Descent found a solution, return it
        return retval, retops
    ops.pop()  # Discard multiplication

    # Descending the tree didn't work, try concatenation
    ops.append("||")
    if concat(curval, val) == target and len(vals) == 0:
        return True, ops  # We have a solution

    # Concatenation didn't work, descend the tree
    retval, retops = test_concat(vals[:], target, concat(curval, val), ops[:])
    if retval:  # Descent found a solution, return it
        return retval, retops
    ops.pop()  # Discard concatenation
    return retval, ops


def test_data_concat(data: list[tuple[int, list[int]]]) -> list[tuple[tuple[bool, lis
    """Returns list of test results and corresponding targets"""
    results = []

    for vals, target in data:
        results.append((test_concat(vals[:], target), target))

    return results


def get_total_concat(data: list[tuple[int, list[int]]]) -> int:
    """Returns sum of targets for equations passing the test"""
    result = test_data_concat(data)
    return sum([int(target) for (retval, target) in result if retval[0]])


data = load_data(test_path)
get_total_concat(data)
```

Out[4]: 11387

In [5]:
```
data = load_data(data_path)
get_total_concat(data)
```

Out[5]: 159490400628354

In [ ]: