Day 15

Part 1

You appear back inside your own mini submarine! Each Historian drives their mini submarine in a different direction; maybe the Chief has his own submarine down here somewhere as well?

You look up to see a vast school of lanternfish swimming past you. On closer inspection, they seem quite anxious, so you drive your mini submarine over to see if you can help.

Because lanternfish populations grow rapidly, they need a lot of food, and that food needs to be stored somewhere. That's why these lanternfish have built elaborate warehouse complexes operated by robots!

These lanternfish seem so anxious because they have lost control of the robot that operates one of their most important warehouses! It is currently running amok, pushing around boxes in the warehouse with no regard for lanternfish logistics or lanternfish inventory management strategies.

Right now, none of the lanternfish are brave enough to swim up to an unpredictable robot so they could shut it off. However, if you could anticipate the robot's movements, maybe they could find a safe option.

The lanternfish already have a map of the warehouse and a list of movements the robot will attempt to make (your puzzle input). The problem is that the movements will sometimes fail as boxes are shifted around, making the actual movements of the robot difficult to predict.

For example:

```
##########
#..O..O.O#
#......O.#
#.OO..O.O#
#..O@..O.#
#O#..O...#
#O..O..O.#
#.OO.O.OO#
#....O...#
##########

<vv>^<v^>v>^vv^v>v<>v^v<v<^vv<<<^><<><>>v<vvv<>^v^>^<<<><<v<<<v^vv^v>^
vvv<<^>^v^^><<>>><>^<<><^vv^^<>vvv<>><^^v>^>vv<>v<<<<v<^v>^<^^>>>^<v<v
><>vv>v^v^<>><>>>><^^>vv>v<^^^>>v^v^<^^>v^^>v^<^v>v<>>v^v^<v>v^^<^^vv<
<<v<^>>^^^^>>>v^<>vvv^><v<<<>^^^vv^<vvv>^>v<^^^^v<>^>vvvv><>>v^<<^^^^^
^><^><>>><>^^<<^^v>>><^<v>^<vv>>v>>>^v><>^v><<<<v>>v<v<v>vvv>^<><<>^><
^>><>^v<><^vvv<^^<><v<<<<<><^v<<<><<<^^<v<^^^><^>>^<v^>o<<<^>>^v<v^v<v^
>^>>^v>vv>^<<^v<>><><<<v<<v><>v<^vv<<<>^^v^>^^>>><<^v>>v^v><^^>>^<>vv^
<><^^>^^^<><vvvvv^v<v<<>^v<v>v<<^><<><<><<<^^<<<^<<>><<><^^^>^^<>^>v<>
^^>vv<^v^v<vv>^<><v<^v>^^^>>>^^vvv^>vvv<>>>^<^>>>>>^<<^v>^vvv<>^<><<v>
v^^>>><<^^<>>^v^<v^vv<>v^<<>^<^v^v><^<<<><<^<v><v<>vv>>v><v^<vv<>v^<<^
```

As the robot (`@`) attempts to move, if there are any boxes (`O`) in the way, the robot will also attempt to push those boxes. However, if this action would cause the robot or a box to move into a wall (`#`), nothing moves instead, including the robot. The initial positions of these are shown on the map at the top of the document the lanternfish gave you.

The rest of the document describes the moves (`^` for up, `v` for down, `<` for left, `>` for right) that the robot will attempt to make, in order. (The moves form a single giant sequence; they are broken into multiple lines just to make copy-pasting easier. Newlines within the move sequence should be ignored.)

Here is a smaller example to get started:

```
########
#..O.O.#
##@.O..#
#...O..#
#.#.O..#
#...O..#
#......#
########

<^^>>>vv<v>>v<<
```

Were the robot to attempt the given sequence of moves, it would push around the boxes as follows:

Initial state:

```
########
#..O.O.#
##@.O..#
#...O..#
#.#.O..#
#...O..#
#......#
########
```

Move `<`:

```
########
#..O.O.#
##@.O..#
#...O..#
#.#.O..#
#...O..#
#......#
########
```

Move `^`:

```
########
#.@O.O.#
##..O..#
#...O..#
#.#.O..#
#...O..#
#......#
########
```

Move `^`:

```
########
#.@O.O.#
##..O..#
```

```
#...O..#
#.#.O..#
#...O..#
#......#
########
```

Move `>`:

```
########
#..@OO.#
##..O..#
#...O..#
#.#.O..#
#...O..#
#......#
########
```

Move `>`:

```
########
#...@OO#
##..O..#
#...O..#
#.#.O..#
#...O..#
#......#
########
```

Move `>`:

```
########
#...@OO#
##..O..#
#...O..#
#.#.O..#
#...O..#
#......#
########
```

Move `v`:

```
########
#....OO#
##..@..#
#...O..#
#.#.O..#
#...O..#
#...O..#
########
```

Move `v`:

```
########
#....OO#
##..@..#
#...O..#
#.#.O..#
#...O..#
```

```
#...O..#
########
```

Move `<`:

```
########
#....OO#
##.@...#
#...O..#
#.#.O..#
#...O..#
#...O..#
########
```

Move `v`:

```
########
#....OO#
##.....#
#..@O..#
#.#.O..#
#...O..#
#...O..#
########
```

Move `>`:

```
########
#....OO#
##.....#
#...@O.#
#.#.O..#
#...O..#
#...O..#
########
```

Move `>`:

```
########
#....OO#
##.....#
#....@O#
#.#.O..#
#...O..#
#...O..#
########
```

Move `v`:

```
########
#....OO#
##.....#
#.....O#
#.#.O@.#
#...O..#
#...O..#
########
```

Move `<`:

```
########
#....OO#
##.....#
#.....O#
#.#O@..#
#...O..#
#...O..#
########
```

Move `<`:

```
########
#....OO#
##.....#
#.....O#
#.#O@..#
#...O..#
#...O..#
########
```

The larger example has many more moves; after the robot has finished those moves, the warehouse would look like this:

```
##########
#.O.O.OOO#
#........#
#OO......#
#OO@.....#
#O#.....O#
#O.....OO#
#O.....OO#
#OO....OO#
##########
```

The lanternfish use their own custom Goods Positioning System (GPS for short) to track the locations of the boxes. The GPS coordinate of a box is equal to `100 times its distance from the top edge of the map plus its distance from the left edge of the map`. (This process does not stop at wall tiles; measure all the way to the edges of the map.)

So, the box shown below has a distance of 1 from the top edge of the map and 4 from the left edge of the map, resulting in a GPS coordinate of `100 * 1 + 4 = 104`.

```
#######
#...O..
#......
```

The lanternfish would like to know the sum of all boxes' GPS coordinates after the robot finishes moving. In the larger example, the sum of all boxes' GPS coordinates is `10092`. In the smaller example, the sum is `2028`.

Predict the motion of the robot and boxes in the warehouse. After the robot is finished moving, what is the sum of all boxes' GPS coordinates?

In [1]:
```
from pathlib import Path

import numpy as np
```

In [2]:
```
test_path_small = Path("data/day15_test_small.txt")
```

```
test_path_large = Path("data/day15_test_large.txt")
data_path = Path("data/day15_data.txt")


def load_data(fpath: Path) -> dict:
    """Returns input data keyed as 'warehouse' and 'commands'.

    The warehouse map is a numpy array, the commands are a string
    """
    data = {}

    with fpath.open() as ifh:
        whs = True
        warehouse, cmds = [], []
        for line in (_.strip() for _ in ifh.readlines()):
            if len(line) == 0:
                whs = False
            elif whs:  # Loading warehouse layout data
                warehouse.append(list(line))
            elif not whs:
                cmds.append(line)

        data["warehouse"] = np.array(warehouse, dtype=str)
        data["commands"] = list("".join(cmds))

    return data


def update_warehouse(data: dict) -> dict:
    """Returns map updated by one step, remaining commands.

    The command that was followed is popped from the list; the
    map is updated with robot/box locations by one step
    """
    whs = data["warehouse"]
    roboloc = [int(_[0]) for _ in np.where(whs == "@")]  # get robot location
    cmd = data["commands"].pop(0)  # get next movement

    # Attempt to move the robot
    if cmd == "<":  # attempt move W
        # If there's a dot W, we can move
        if whs[roboloc[0]][roboloc[1]-1] == ".":
            whs[roboloc[0]][roboloc[1]-1] = "@"
            whs[roboloc[0]][roboloc[1]] = "."
        elif whs[roboloc[0]][roboloc[1]-1] == "O":  # There's a box
            # How many boxes to the W?
            boxcount = 1
            while whs[roboloc[0]][roboloc[1]-boxcount] == "O":
                boxcount += 1
            boxcount -= 1
            # Is there a space to move the boxes?
            if whs[roboloc[0]][roboloc[1]-(boxcount+1)] == ".":
                whs[roboloc[0]][roboloc[1]-(boxcount+1)] = "O"  # move box to end
                whs[roboloc[0]][roboloc[1]-1] = "@"  # move robot
                whs[roboloc[0]][roboloc[1]] = "."
    elif cmd == ">":  # attempt move E
        # If there's a dot E, we can move
        if whs[roboloc[0]][roboloc[1]+1] == ".":
            whs[roboloc[0]][roboloc[1]+1] = "@"
            whs[roboloc[0]][roboloc[1]] = "."
        elif whs[roboloc[0]][roboloc[1]+1] == "O":  # There's a box
            # How many boxes to the E?
            boxcount = 1
            while whs[roboloc[0]][roboloc[1]+boxcount] == "O":
                boxcount += 1
```

```
            boxcount -= 1
            # Is there a space to move the boxes?
            if whs[roboloc[0]][roboloc[1]+(boxcount+1)] == ".":
                whs[roboloc[0]][roboloc[1]+(boxcount+1)] = "O"  # move box to end
                whs[roboloc[0]][roboloc[1]+1] = "@"  # move robot
                whs[roboloc[0]][roboloc[1]] = "."
    elif cmd == "^":  # attempt move N
        # If there's a dot N, we can move
        if whs[roboloc[0]-1][roboloc[1]] == ".":
            whs[roboloc[0]-1][roboloc[1]] = "@"
            whs[roboloc[0]][roboloc[1]] = "."
        elif whs[roboloc[0]-1][roboloc[1]] == "O":  # There's a box
            # How many boxes to the N?
            boxcount = 1
            while whs[roboloc[0]-boxcount][roboloc[1]] == "O":
                boxcount += 1
            boxcount -= 1
            # Is there a space to move the boxes?
            if whs[roboloc[0]-(boxcount+1)][roboloc[1]] == ".":
                whs[roboloc[0]-(boxcount+1)][roboloc[1]] = "O"  # move box to end
                whs[roboloc[0]-1][roboloc[1]] = "@"  # move robot
                whs[roboloc[0]][roboloc[1]] = "."
    elif cmd == "v":  # attempt move S
        # If there's a dot S, we can move
        if whs[roboloc[0]+1][roboloc[1]] == ".":
            whs[roboloc[0]+1][roboloc[1]] = "@"
            whs[roboloc[0]][roboloc[1]] = "."
        elif whs[roboloc[0]+1][roboloc[1]] == "O":  # There's a box
            # How many boxes to the E?
            boxcount = 1
            while whs[roboloc[0]+boxcount][roboloc[1]] == "O":
                boxcount += 1
            boxcount -= 1
            # Is there a space to move the boxes?
            if whs[roboloc[0]+(boxcount+1)][roboloc[1]] == ".":
                whs[roboloc[0]+(boxcount+1)][roboloc[1]] = "O"  # move box to end
                whs[roboloc[0]+1][roboloc[1]] = "@"  # move robot
                whs[roboloc[0]][roboloc[1]] = "."

    return data


def score_warehouse(warehouse: np.array) -> int:
    """Returns the score of all box locations in the warehouse."""
    boxes = np.where(warehouse == "O")
    return int(sum(boxes[0] * 100) + sum(boxes[1]))


data = load_data(test_path_small)
while len(data["commands"]):
    update_warehouse(data)
score_warehouse(data["warehouse"])
```

Out[2]: 2028

In [3]:
```
data = load_data(test_path_large)
while len(data["commands"]):
    update_warehouse(data)
score_warehouse(data["warehouse"])
```

Out[3]: 10092

In [4]:
```
data = load_data(data_path)
while len(data["commands"]):
```

```
    update_warehouse(data)
score_warehouse(data["warehouse"])
```

Out[4]: 1448589

Part 2

The lanternfish use your information to find a safe moment to swim in and turn off the malfunctioning robot! Just as they start preparing a festival in your honor, reports start coming in that a second warehouse's robot is also malfunctioning.

This warehouse's layout is surprisingly similar to the one you just helped. There is one key difference: everything except the robot is twice as wide! The robot's list of movements doesn't change.

To get the wider warehouse's map, start with your original map and, for each tile, make the following changes:

- If the tile is #, the new map contains ## instead.
- If the tile is O, the new map contains [] instead.
- If the tile is ., the new map contains .. instead.
- If the tile is @, the new map contains @. instead.

This will produce a new warehouse map which is twice as wide and with wide boxes that are represented by []. (The robot does not change size.)

The larger example from before would now look like this:

```
####################
##....[]....[]..[]##
##............[]..##
##..[][]....[]..[]##
##....[]@.....[]..##
##[]##....[]......##
##[]....[]....[]..##
##..[][]..[]..[][]##
##........[]......##
####################
```

Because boxes are now twice as wide but the robot is still the same size and speed, boxes can be aligned such that they directly push two other boxes at once. For example, consider this situation:

```
#######
#...#.#
#.....#
#..OO@#
#..O..#
#.....#
#######

<vv<<^^<<^^
```

After appropriately resizing this map, the robot would push around these boxes as follows:

Initial state:

```
##############
##......##..##
##..........##
##....[][]@.##
##....[]....##
##..........##
##############
```

Move `<`:

```
##############
##......##..##
##..........##
##...[][]@..##
##....[]....##
##..........##
##############
```

Move `v`:

```
##############
##......##..##
##..........##
##...[][]...##
##....[].@..##
##..........##
##############
```

Move `v`:

```
##############
##......##..##
##..........##
##...[][]...##
##....[]....##
##.......@..##
##############
```

Move `<`:

```
##############
##......##..##
##..........##
##...[][]...##
##....[]....##
##......@...##
##############
```

Move `<`:

```
##############
##......##..##
##..........##
##...[][]...##
##....[]....##
##.....@....##
##############
```

Move `^`:

```
##############
##......##..##
##...[][]...##
##....[]....##
##.....@....##
##..........##
##############
```

Move `^`:

```
##############
##......##..##
##...[][]...##
##....[]....##
##.....@....##
##..........##
##############
```

Move `<`:

```
##############
##......##..##
##...[][]...##
##....[]....##
##....@.....##
##..........##
##############
```

Move `<`:

```
##############
##......##..##
##...[][]...##
##....[]....##
##...@......##
##..........##
##############
```

Move `^`:

```
##############
##......##..##
##...[][]...##
##...@[]....##
##..........##
##..........##
##############
```

Move `^`:

```
##############
##...[].##..##
##...@.[]...##
##....[]....##
##..........##
##..........##
##############
```

This warehouse also uses GPS to locate the boxes. For these larger boxes, distances are measured from the edge of the map to the closest edge of the box in question. So, the box shown below has a distance of 1 from the top edge of the map and 5 from the left edge of the map, resulting in a GPS coordinate of 100 * 1 + 5 = 105.

```
##########
##...[]...
##........
```

In the scaled-up version of the larger example from above, after the robot has finished all of its moves, the warehouse would look like this:

```
####################
##[].......[].[][]##
##[]...........[].##
##[]........[][][]##
##[]......[]....[]##
##..##......[]....##
##..[]............##
##..@......[].[][]##
##......[][]..[]..##
####################
```

The sum of these boxes' GPS coordinates is 9021.

Predict the motion of the robot and boxes in this new, scaled-up warehouse. What is the sum of all boxes' final GPS coordinates?

In [5]:
```python
test_path_wide_small = Path("data/day15_test_wide_small.txt")


def load_wide_data(fpath: Path) -> dict:
    """Returns dict of warehouse layout and commands.

    Warehouse is an np.array keyed as "warehouse", "commands" keys
    for the commands as a string.
    """
    data = {}

    with fpath.open() as ifh:
        whs = True
        warehouse, cmds = [], []
        for line in (_.strip() for _ in ifh.readlines()):
            if len(line) == 0:
                whs = False
            elif whs:  # Loading warehouse layout data
                line = line.replace("#", "##").replace("O", "[]").replace(".", "..").
                warehouse.append(list(line))
            elif not whs:
                cmds.append(line)

        data["warehouse"] = np.array(warehouse, dtype=str)
        data["commands"] = list("".join(cmds))

        return data


def update_wide_warehouse(data: dict) -> dict:
    """Returns warehouse layout updated with next command.

    Warehouse is updated in place, used command is popped from the string.
```

```
    """
    whs = data["warehouse"]
    roboloc = [int(_[0]) for _ in np.where(whs == "@")]  # get robot location
    cmd = data["commands"].pop(0)  # get next movement

    if cmd == "<":  # attempt move W
        # If there's a dot W, we can move
        if whs[roboloc[0]][roboloc[1]-1] == ".":
            whs[roboloc[0]][roboloc[1]-1] = "@"
            whs[roboloc[0]][roboloc[1]] = "."
        elif whs[roboloc[0]][roboloc[1]-1] in "[]":  # There's a box
            # How many boxes to the W?
            boxcount = 1
            while whs[roboloc[0]][roboloc[1]-boxcount] in "[]":
                boxcount += 1
            boxcount -= 1
            # Is there a space to move the boxes?
            if whs[roboloc[0]][roboloc[1]-(boxcount+1)] == ".":
                for pos in range(roboloc[1]-(boxcount+1), roboloc[1]):
                    whs[roboloc[0]][pos] = whs[roboloc[0]][pos+1]  # move boxes, robo
                whs[roboloc[0]][roboloc[1]] = "."
    elif cmd == ">":  # attempt move E
        # If there's a dot E, we can move
        if whs[roboloc[0]][roboloc[1]+1] == ".":
            whs[roboloc[0]][roboloc[1]+1] = "@"
            whs[roboloc[0]][roboloc[1]] = "."
        elif whs[roboloc[0]][roboloc[1]+1] in "[]":  # There's a box
            # How many boxes to the E?
            boxcount = 1
            while whs[roboloc[0]][roboloc[1]+boxcount] in "[]":
                boxcount += 1
            boxcount -= 1
            # Is there a space to move the boxes?
            if whs[roboloc[0]][roboloc[1]+(boxcount+1)] == ".":
                for pos in range(roboloc[1]+(boxcount+1), roboloc[1], -1):
                    whs[roboloc[0]][pos] = whs[roboloc[0]][pos-1]  # move boxes, robo
                whs[roboloc[0]][roboloc[1]] = "."
    elif cmd == "^":  # attempt move N
        boxes = []  # Boxes that will be moved
        # If there's a dot N, we can move
        if whs[roboloc[0]-1][roboloc[1]] == ".":
            whs[roboloc[0]-1][roboloc[1]] = "@"
            whs[roboloc[0]][roboloc[1]] = "."
        # Is there a box that can be pushed?
        elif "]" == whs[roboloc[0]-1][roboloc[1]]:  # pushing right side
            boxes.append((roboloc[0]-1, roboloc[1]-1))  # store "[" location
        elif "[" == whs[roboloc[0]-1][roboloc[1]]:  # pushing left side direct
            boxes.append((roboloc[0]-1, roboloc[1]))
        # Loop over all boxes and identify the cascade that will be pushed
        hit_wall = False
        if len(boxes):
            boxcount = len(boxes)
            while True:
                lastlen = len(boxes)
                for ybox, xbox in boxes:
                    if "]" == whs[ybox-1][xbox]:  # pushing right side
                        if (ybox-1, xbox-1) not in boxes:
                            boxes.append((ybox-1, xbox-1))
                    if "[" == whs[ybox-1][xbox]:  # pushing left side direct
                        if (ybox-1, xbox) not in boxes:
                            boxes.append((ybox-1, xbox))
                    if "[" == whs[ybox-1][xbox+1]:  # pushing left side indirect
                        if (ybox-1, xbox+1) not in boxes:
                            boxes.append((ybox-1, xbox+1))
                    if "#" == whs[ybox-1][xbox] or "#" == whs[ybox-1][xbox+1]:  # hit
```

```
                    hit_wall = True
                if lastlen == len(boxes) or hit_wall is True:
                    break
            if hit_wall is False:  # We can move the box cascade
                while len(boxes):
                    ybox, xbox = boxes.pop(-1)
                    whs[ybox-1][xbox] = "["
                    whs[ybox-1][xbox+1] = "]"
                    whs[ybox][xbox] = "."
                    whs[ybox][xbox+1] = "."
                # Move the robot
                whs[roboloc[0]-1][roboloc[1]] = "@"
                whs[roboloc[0]][roboloc[1]] = "."
    elif cmd == "v":  # attempt move S
        boxes = []  # Boxes that will be moved
        # If there's a dot S, we can move
        if whs[roboloc[0]+1][roboloc[1]] == ".":
            whs[roboloc[0]+1][roboloc[1]] = "@"
            whs[roboloc[0]][roboloc[1]] = "."
        # Is there a box that can be pushed?
        elif "]" == whs[roboloc[0]+1][roboloc[1]]:  # pushing right side
            boxes.append((roboloc[0]+1, roboloc[1]-1))  # store "[" location
        elif "[" == whs[roboloc[0]+1][roboloc[1]]:  # pushing left side direct
            boxes.append((roboloc[0]+1, roboloc[1]))
        # Loop over all boxes and identify the cascade that will be pushed
        hit_wall = False
        if len(boxes):
            boxcount = len(boxes)
            while True:
                lastlen = len(boxes)
                for ybox, xbox in boxes:
                    if "]" == whs[ybox+1][xbox]:  # pushing right side
                        if (ybox+1, xbox-1) not in boxes:
                            boxes.append((ybox+1, xbox-1))
                    if "[" == whs[ybox+1][xbox]:  # pushing left side direct
                        if (ybox+1, xbox) not in boxes:
                            boxes.append((ybox+1, xbox))
                    if "[" == whs[ybox+1][xbox+1]:  # pushing left side indirect
                        if (ybox+1, xbox+1) not in boxes:
                            boxes.append((ybox+1, xbox+1))
                    if "#" == whs[ybox+1][xbox] or "#" == whs[ybox+1][xbox+1]:  # hit
                        hit_wall = True
                if lastlen == len(boxes) or hit_wall is True:
                    break
            if hit_wall is False:  # We can move the box cascade
                while len(boxes):
                    ybox, xbox = boxes.pop(-1)
                    whs[ybox+1][xbox] = "["
                    whs[ybox+1][xbox+1] = "]"
                    whs[ybox][xbox] = "."
                    whs[ybox][xbox+1] = "."
                # Move the robot
                whs[roboloc[0]+1][roboloc[1]] = "@"
                whs[roboloc[0]][roboloc[1]] = "."

    return data


def score_wide_warehouse(warehouse: np.array) -> int:
    """Returns the score of all box locations in the warehouse."""
    boxes = np.where(warehouse == "[")
    return int(sum(boxes[0] * 100) + sum(boxes[1]))


data = load_wide_data(test_path_wide_small)
```

```
while len(data["commands"]):
    data = update_wide_warehouse(data)


data = load_wide_data(test_path_large)
while len(data["commands"]):
    update_wide_warehouse(data)
score_wide_warehouse(data["warehouse"])
```

Out[5]: 9021

In [6]:
```
data = load_wide_data(data_path)
while len(data["commands"]):
    update_wide_warehouse(data)
score_wide_warehouse(data["warehouse"])
```

Out[6]: 1472235

In [ ]: