Day 6

Part 1

The Historians use their fancy device again, this time to whisk you all away to the North Pole prototype suit manufacturing lab... in the year 1518! It turns out that having direct access to history is very convenient for a group of historians.

You still have to be careful of time paradoxes, and so it will be important to avoid anyone from 1518 while The Historians search for the Chief. Unfortunately, a single guard is patrolling this part of the lab.

Maybe you can work out where the guard will go ahead of time so that The Historians can search safely?

You start by making a map (your puzzle input) of the situation. For example:

```
....#.....
.........#
..........
..#.......
.......#..
..........
.#..^.....
........#.
#.........
......#...
```

The map shows the current position of the guard with ^ (to indicate the guard is currently facing up from the perspective of the map). Any obstructions - crates, desks, alchemical reactors, etc. - are shown as #.

Lab guards in 1518 follow a very strict patrol protocol which involves repeatedly following these steps:

```
If there is something directly in front of you, turn right 90 degrees.
Otherwise, take a step forward.
```

Following the above protocol, the guard moves up several times until she reaches an obstacle (in this case, a pile of failed suit prototypes):

```
....#.....
....^....#
..........
..#.......
.......#..
..........
.#........
........#.
#.........
......#...
```

Because there is now an obstacle in front of the guard, she turns right before continuing straight in her new facing direction:

```
....#.....
.........>#
..........
..#.......
.......#..
..........
.#........
........#.
#.........
......#...
```

Reaching another obstacle (a spool of several very long polymers), she turns right again and continues downward:

```
....#.....
.........#
..........
..#.......
.......#..
..........
.#......v.
........#.
#.........
......#...
```

This process continues for a while, but the guard eventually leaves the mapped area (after walking past a tank of universal solvent):

```
....#.....
.........#
..........
..#.......
.......#..
..........
.#........
........#.
#.........
......#v..
```

By predicting the guard's route, you can determine which specific positions in the lab will be in the patrol path. Including the guard's starting position, the positions visited by the guard before leaving the area are marked with an X:

```
....#.....
....XXXXX#
....X...X.
..#.X...X.
..XXXXX#X.
..X.X.X.X.
.#XXXXXXX.
.XXXXXXX#.
#XXXXXXX..
......#X..
```

In this example, the guard will visit `41` distinct positions on your map.

Predict the path of the guard. How many distinct positions will the guard visit before leaving the mapped area?

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

import numpy as np
```

In [2]:
```
test_path = Path("data/day06_test.txt")
data_path = Path("data/day06_data.txt")

def load_data(fpath: Path) -> tuple[np.array, tuple[int, int]]:
    """Returns map as an array, and the guard start position"""
    labmap = []

    for line in fpath.open().readlines():
        # 0 for space, 1 for block, 2 for guard start
        labmap.append([0 if _ == "." else (1 if _ == "#" else 2) for _ in line.strip(

    labmap = np.array(labmap)
    start = np.where(labmap == 2)

    return labmap, (int(start[0][0]), int(start[1][0]))


def next_steps(xstep, ystep):
    if (xstep, ystep) == (0, -1):
        return (1, 0)
    elif (xstep, ystep) == (1, 0):
        return (0, 1)
    elif (xstep, ystep) == (0, 1):
        return (-1, 0)
    elif (xstep, ystep) == (-1, 0):
        return (0, -1)


def make_guard_visits(labmap, start, xstep=0, ystep=-1):
    # Get map limits
    xlim, ylim = labmap.shape

    # Place guard
    guard_pos = start

    # Positions already visited by guard
    seen = set()

    while True:
        # Mark the map
        seen.add(guard_pos)

        # If the guard would step off the map, return the count of locations
        if not (ylim > guard_pos[0] + ystep >= 0 and xlim > guard_pos[1] + xstep >= 0
            return len(seen)

        # If the guard would hit an obstacle rotate 90 degrees clockwise
        if labmap[guard_pos[0] + ystep][guard_pos[1] + xstep] == 1:  # Turn 90 degree
            xstep, ystep = next_steps(xstep, ystep)
        else:  # Guard steps forward
            guard_pos = (guard_pos[0] + ystep, guard_pos[1] + xstep)


labmap, start = load_data(test_path)
make_guard_visits(labmap, start)
```

Out[2]: 41

In [3]:
```
labmap, start = load_data(data_path)
make_guard_visits(labmap, start)
```

Out[3]: 5080

Part 2

While The Historians begin working around the guard's patrol route, you borrow their fancy device and step outside the lab. From the safety of a supply closet, you time travel through the last few months and record the nightly status of the lab's guard post on the walls of the closet.

Returning after what seems like only a few seconds to The Historians, they explain that the guard's patrol area is simply too large for them to safely search the lab without getting caught.

Fortunately, they are pretty sure that adding a single new obstruction won't cause a time paradox. They'd like to place the new obstruction in such a way that the guard will get stuck in a loop, making the rest of the lab safe to search.

To have the lowest chance of creating a time paradox, The Historians would like to know all of the possible positions for such an obstruction. The new obstruction can't be placed at the guard's starting position - the guard is there right now and would notice.

In the above example, there are only 6 different positions where a new obstruction would cause the guard to get stuck in a loop. The diagrams of these six situations use O to mark the new obstruction, | to show a position where the guard moves up/down, - to show a position where the guard moves left/right, and + to show a position where the guard moves both up/down and left/right.

Option one, put a printing press next to the guard's starting position:

```
....#.....
....+---+#
....|...|.
..#.|...|.
....|..#|.
....|...|.
.#.O^---+.
........#.
#.........
......#...
```

Option two, put a stack of failed suit prototypes in the bottom right quadrant of the mapped area:

```
....#.....
....+---+#
....|...|.
..#.|...|.
..+-+-+#|.
..|.|.|.|.
.#+-^-+-+.
......O.#.
#.........
......#...
```

Option three, put a crate of chimney-squeeze prototype fabric next to the standing desk in the bottom right quadrant:

```
....#.....
....+---+#
....|...|.
..#.|...|.
..+-+-+#|.
..|.|.|.|.
.#+-^-+-+.
.+----+0#.
#+----+...
......#...
```

Option four, put an alchemical retroencabulator near the bottom left corner:

```
....#.....
....+---+#
....|...|.
..#.|...|.
..+-+-+#|.
..|.|.|.|.
.#+-^-+-+.
..|...|.#.
#0+---+...
......#...
```

Option five, put the alchemical retroencabulator a bit to the right instead:

```
....#.....
....+---+#
....|...|.
..#.|...|.
..+-+-+#|.
..|.|.|.|.
.#+-^-+-+.
....|.|.#.
#..0+-+...
......#...
```

Option six, put a tank of sovereign glue right next to the tank of universal solvent:

```
....#.....
....+---+#
....|...|.
..#.|...|.
..+-+-+#|.
..|.|.|.|.
.#+-^-+-+.
.+----++#.
#+----++..
......#0..
```

It doesn't really matter what you choose to use as an obstacle so long as you and The Historians can put it into position without the guard noticing. The important thing is having enough options that you can find one that minimizes time paradoxes, and in this example, there are 6 different positions you could choose.

You need to get the guard stuck in a loop by adding a single new obstruction. How many different positions could you choose for this obstruction?

In [4]:
```
def check_loop(labmap, start, xstep=0, ystep=-1):
```

```
    """Returns True if the guard would stick in a loop"""
    # Get map limits
    xlim, ylim = labmap.shape

    # Place guard
    guard_pos = start

    # Locations visited (with direction)
    seen = set()

    while True:
        # If we've seen the current position/direction before, we're in a loop
        if (guard_pos, xstep, ystep) in seen:
            return True
        seen.add((guard_pos, xstep, ystep))  # Otherwise record where we are

        # If the guard would step off the map, return False
        if not (ylim > guard_pos[0] + ystep >= 0 and xlim > guard_pos[1] + xstep >= 0
            return False

        # If the guard would hit an obstacle rotate 90 degrees clockwise
        if labmap[guard_pos[0] + ystep][guard_pos[1] + xstep] == 1:  # Turn 90 degree
            xstep, ystep = next_steps(xstep, ystep)
        else:  # Otherwise, the guard steps forward
            guard_pos = (guard_pos[0] + ystep, guard_pos[1] + xstep)


def count_obstacles(labmap, start, xstep=0, ystep=-1):
    # Get map limits
    xlim, ylim = labmap.shape

    # Place guard
    guard_pos = start
    startxstep, startystep = xstep, ystep

    # Obstacle locations
    obstacles = set()

    while True:
        # If the guard would step off the map, return the count of added obstacles
        if not (ylim > guard_pos[0] + ystep >= 0 and xlim > guard_pos[1] + xstep >= 0
            return len(obstacles)

        # If the guard would hit an obstacle rotate 90 degrees clockwise
        if labmap[guard_pos[0] + ystep][guard_pos[1] + xstep] == 1:  # Turn 90 degree
            xstep, ystep = next_steps(xstep, ystep)
        else:  # If we put an obstacle in front of the guard, would we cause a loop?
            # Add the obstacle on a new map
            newmap = np.copy(labmap)
            obsrow, obscol = guard_pos[0] + ystep, guard_pos[1] + xstep
            newmap[obsrow][obscol] = 1
            # Check for an infinite loop - if there is one, keep the obstacle
            if check_loop(newmap, start, startxstep, startystep):
                obstacles.add((obsrow, obscol))
            # Guard steps forward
            guard_pos = (guard_pos[0] + ystep, guard_pos[1] + xstep)


labmap, start = load_data(test_path)
count_obstacles(labmap, start)
```

Out[4]: 6

In [5]:
```
labmap, start = load_data(data_path)
count_obstacles(labmap, start)
```

Out[5]: 1919

In [ ]: