Day 8

Part 1

You find yourselves on the roof of a top-secret Easter Bunny installation.

While The Historians do their thing, you take a look at the familiar huge antenna. Much to your surprise, it seems to have been reconfigured to emit a signal that makes people 0.1% more likely to buy Easter Bunny brand Imitation Mediocre Chocolate as a Christmas gift! Unthinkable!

Scanning across the city, you find that there are actually many such antennas. Each antenna is tuned to a specific frequency indicated by a single lowercase letter, uppercase letter, or digit. You create a map (your puzzle input) of these antennas. For example:

```
............
........0...
.....0......
.......0....
....0.......
......A.....
............
............
........A...
.........A..
............
............
```

The signal only applies its nefarious effect at specific antinodes based on the resonant frequencies of the antennas. In particular, an antinode occurs at any point that is perfectly in line with two antennas of the same frequency - but only when one of the antennas is twice as far away as the other. This means that for any pair of antennas with the same frequency, there are two antinodes, one on either side of them.

So, for these two antennas with frequency `a`, they create the two antinodes marked with `#`:

```
..........
...#......
..........
....a.....
..........
.....a....
..........
......#...
..........
..........
```

Adding a third antenna with the same frequency creates several more antinodes. It would ideally add four antinodes, but two are off the right side of the map, so instead it adds only two:

```
..........
...#......
#.........
....a.....
........a.
```

```
.....a....
..#.......
......#...
..........
..........
```

Antennas with different frequencies don't create antinodes; `A` and `a` count as different frequencies. However, antinodes can occur at locations that contain antennas. In this diagram, the lone antenna with frequency capital `A` creates no antinodes but has a lowercase-`a`-frequency antinode at its location:

```
..........
...#......
#.........
....a.....
........a.
.....a....
..#.......
......A...
..........
..........
```

The first example has antennas with two different frequencies, so the antinodes they create look like this, plus an antinode overlapping the topmost `A`-frequency antenna:

```
......#....#
...#....0...
....#0....#.
..#....0....
....0....#..
.#....A.....
...#........
#......#....
........A...
.........A..
..........#.
..........#.
```

Because the topmost `A`-frequency antenna overlaps with a `0`-frequency antinode, there are `14` total unique locations that contain an antinode within the bounds of the map.

Calculate the impact of the signal. How many unique locations within the bounds of the map contain an antinode?

In [1]:
```
from collections import defaultdict
from itertools import combinations
from pathlib import Path

import numpy as np
```

In [2]:
```
test_path = Path("data/day08_test.txt")
data_path = Path("data/day08_data.txt")


def load_data(fpath: Path) -> tuple[np.array, np.array]:
    """Return maps of antennae and antinodes.

    The antinode map is initially empty
    """
```

```
    antmap = []

    with fpath.open() as ifh:
        for line in ifh.readlines():
            antmap.append(list(line.strip()))
    antmap = np.array(antmap)

    return antmap, np.full(antmap.shape, ".", dtype=str)


def locate_antennae(antmap: np.array) -> dict[tuple[int,int]]:
    """Returns dictionary of antenna locations.

    Dictionary is keyed by antenna ID, with a list of co-ordinates for
    antennae with that ID.
    """
    antennae = defaultdict(list)
    nrows, ncols = antmap.shape

    for row in range(nrows):
        for col in range(ncols):
            if antmap[row][col] != ".":
                antennae[antmap[row][col]].append((row, col))

    return antennae


def find_antinodes(antennae: dict[tuple[int,int]], nodemap: np.array) -> np.array:
    """Populates an antinode map with antinodes.

    The map is an array, with "#" where antinodes occur.
    """
    antinodes = set()  # Only keep unique antinodes

    for key, vals in antennae.items():
        antpairs = combinations(vals, 2)
        for ant1, ant2 in antpairs:
            ydist = ant1[0] - ant2[0]
            xdist = ant1[1] - ant2[1]
            antinodes = antinodes.union(set(((ant1[0] + ydist, ant1[1] + xdist),
                                             (ant2[0] - ydist, ant2[1] - xdist))))

    nrows, ncols = nodemap.shape
    # Add antinodes to map, filtering out off-map locations
    for row, col in antinodes:
        if 0 <= row < nrows and 0 <= col < ncols:
            nodemap[row][col] = "#"

    return nodemap

def count_antinodes(antmap: np.array) -> int:
    """Returns count of antinodes in a map."""
    return np.count_nonzero(antmap == "#")


antmap, antinodes = load_data(test_path)
antennae = locate_antennae(antmap)
antinodes = find_antinodes(antennae, antinodes)
count_antinodes(antinodes)
```

Out[2]: 14

In [3]:
```
antmap, antinodes = load_data(data_path)
antennae = locate_antennae(antmap)
```

```
antinodes = find_antinodes(antennae, antinodes)
count_antinodes(antinodes)
```

Out[3]: 379

Part 2

Watching over your shoulder as you work, one of The Historians asks if you took the effects of resonant harmonics into your calculations.

Whoops!

After updating your model, it turns out that an antinode occurs at any grid position exactly in line with at least two antennas of the same frequency, regardless of distance. This means that some of the new antinodes will occur at the position of each antenna (unless that antenna is the only one of its frequency).

So, these three `T`-frequency antennas now create many antinodes:

```
T....#....
...T......
.T....#...
.........#
..#.......
..........
...#......
..........
....#.....
..........
```

In fact, the three `T`-frequency antennas are all exactly in line with two antennas, so they are all also antinodes! This brings the total number of antinodes in the above example to `9`.

The original example now has `34` antinodes, including the antinodes that appear on every antenna:

```
##....#....#
.#.#....0...
..#.#0....#.
..##...0....
....0....#..
.#...#A....#
...#..#.....
#....#.#....
..#.....A...
....#....A..
.#........#.
...#......##
```

Calculate the impact of the signal using this updated model. How many unique locations within the bounds of the map contain an antinode?

In [4]:
```
def find_resonances(antennae: dict[tuple[int,int]], nodemap: np.array) -> np.array:
    """Populates an antinode map with resonant antinodes.

    The map is an array, with "#" where antinodes occur.
    """
    antinodes = set()
    nrows, ncols = nodemap.shape
```

```
    for key, vals in antennae.items():
        antpairs = combinations(vals, 2)
        for ant1, ant2 in antpairs:
            ydist = ant1[0] - ant2[0]
            xdist = ant1[1] - ant2[1]

            # Find resonances from each antenna in turn, until we go off-map.
            # Resonances occur at whole-number multiples of the distance
            # between antennae.
            idx = 0  # This includes the antenna location itself
            while 0 <= ant1[0] + idx * ydist < nrows and 0 <= ant1[1] + idx * xdist <
                antinodes.add((ant1[0] + idx * ydist, ant1[1] + idx * xdist))
                idx += 1
            idx = 0
            while 0 <= ant2[0] - idx * ydist < nrows and 0 <= ant2[1] - idx * xdist <
                antinodes.add((ant2[0] - idx * ydist, ant2[1] - idx * xdist))
                idx += 1

    # Add the antinodes to the map
    nrows, ncols = nodemap.shape
    for row, col in antinodes:
            nodemap[row][col] = "#"

    return nodemap


antmap, antinodes = load_data(test_path)
antennae = locate_antennae(antmap)
antinodes = find_resonances(antennae, antinodes)
count_antinodes(antinodes)
```

Out[4]: 34

In [5]:
```
antmap, antinodes = load_data(data_path)
antennae = locate_antennae(antmap)
antinodes = find_resonances(antennae, antinodes)
count_antinodes(antinodes)
```

Out[5]: 1339

In [ ]: