--- Day 12: Hot Springs ---

You finally reach the hot springs! You can see steam rising from secluded areas attached to the primary, ornate building.

As you turn to enter, the researcher stops you. "Wait - I thought you were looking for the hot springs, weren't you?" You indicate that this definitely looks like hot springs to you.

"Oh, sorry, common mistake! This is actually the onsen! The hot springs are next door."

You look in the direction the researcher is pointing and suddenly notice the massive metal helixes towering overhead. "This way!"

It only takes you a few more steps to reach the main gate of the massive fenced-off area containing the springs. You go through the gate and into a small administrative building.

"Hello! What brings you to the hot springs today? Sorry they're not very hot right now; we're having a **lava shortage** at the moment." You ask about the missing machine parts for Desert Island.

"Oh, all of Gear Island is currently offline! Nothing is being manufactured at the moment, not until we get more lava to heat our forges. And our springs. The springs aren't very springy unless they're hot!"

"Say, could you go up and see why the lava stopped flowing? The springs are too cold for normal operation, but we should be able to find one springy enough to launch **you** up there!"

There's just one problem - many of the springs have fallen into disrepair, so they're not actually sure which springs would even be **safe** to use! Worse yet, their **condition records of which springs are damaged** (your puzzle input) are also damaged! You'll need to help them repair the damaged records.

In the giant field just outside, the springs are arranged into **rows**. For each row, the condition records show every spring and whether it is **operational** (`.`) or **damaged** (`#`). This is the part of the condition records that is itself damaged; for some springs, it is simply **unknown** (`?`) whether the spring is operational or damaged.

However, the engineer that produced the condition records also duplicated some of this information in a different format! After the list of springs for a given row, the size of each **contiguous group of damaged springs** is listed in the order those groups appear in the row. This list always accounts for every damaged spring, and each number is the entire size of its contiguous group (that is, groups are always separated by at least one operational spring: `####` would always be `4`, never `2,2`).

So, condition records with no unknown spring conditions might look like this:

```
#.#.### 1,1,3
.#...#....###. 1,1,3
.#.###.#.###### 1,3,1,6
####.#...#... 4,1,1
#....######..#####. 1,6,5
.###.##....# 3,2,1
```

However, the condition records are partially damaged; some of the springs' conditions are actually **unknown** (`?`). For example:

```
???.### 1,1,3
.??..??...?##. 1,1,3
?#?#?#?#?#?#?#? 1,3,1,6
????.#...#... 4,1,1
????.######..#####. 1,6,5
?###???????? 3,2,1
```

Equipped with this information, it is your job to figure out **how many different arrangements** of operational and broken springs fit the given criteria in each row.

In the first line (`???.### 1,1,3`), there is exactly **one** way separate groups of one, one, and three broken springs (in that order) can appear in that row: the first three unknown springs must be broken, then operational, then broken (`#.#`), making the whole row `#.#.###`.

The second line is more interesting: `.??..??...?##. 1,1,3` could be a total of **four** different arrangements. The last `?` must always be broken (to satisfy the final contiguous group of three broken springs), and each `??` must hide exactly one of the two broken springs. (Neither `??` could be both broken springs or they would form a single contiguous group of two; if that were true, the numbers afterward would have been `2,3` instead.) Since each `??` can either be `#.` or `.#`, there are four possible arrangements of springs.

The last line is actually consistent with **ten** different arrangements! Because the first number is `3`, the first and second `?` must both be `.` (if either were `#`, the first number would have to be `4` or higher). However, the remaining run of unknown spring conditions have many different ways they could hold groups of two and one broken springs:

```
?###???????? 3,2,1
.###.##.#...
.###.##..#..
.###.##...#.
.###.##....#
.###..##.#..
.###..##..#.
.###..##...#
.###...##.#.
.###...##..#
.###....##.#
```

In this example, the number of possible arrangements for each row is:

- `???.### 1,1,3` - **1** arrangement
- `.??..??...?##. 1,1,3` - **4** arrangements
- `?#?#?#?#?#?#?#? 1,3,1,6` - **1** arrangement
- `????.#...#... 4,1,1` - **1** arrangement
- `????.######..#####. 1,6,5` - **4** arrangements
- `?###???????? 3,2,1` - **10** arrangements

Adding all of the possible arrangement counts together produces a total of **21** arrangements.

For each row, count all of the different arrangements of operational and broken springs that meet the given criteria. **What is the sum of those counts?**

Your puzzle answer was `7599`.

The first half of this puzzle is complete! It provides one gold star: *

--- Part Two ---

As you look out at the field of springs, you feel like there are way more springs than the condition records list. When you examine the records, you discover that they were actually **folded up** this whole time!

To **unfold the records**, on each row, replace the list of spring conditions with five copies of itself (separated by `?`) and replace the list of contiguous groups of damaged springs with five copies of itself (separated by `,`).

So, this row:

```
.# 1
```

Would become:

```
.#?.#?.#?.#?.# 1,1,1,1,1
```

The first line of the above example would become:

```
???.###????.###????.###????.###????.### 1,1,3,1,1,3,1,1,3,1,1,3,1,1,3
```

In the above example, after unfolding, the number of possible arrangements for some rows is now much larger:

- `???.### 1,1,3` - **1** arrangement
- `.??..??...?##. 1,1,3` - **16384** arrangements
- `?#?#?#?#?#?#?#? 1,3,1,6` - **1** arrangement
- `????.#...#... 4,1,1` - **16** arrangements
- `????.######..#####. 1,6,5` - **2500** arrangements
- `?###???????? 3,2,1` - **506250** arrangements

After unfolding, adding all of the possible arrangement counts together produces **525152**.

Unfold your condition records; **what is the new sum of possible arrangement counts?**

Answer: [Submit]

Although it hasn't changed, you can still get your puzzle input.

You can also [Share] this puzzle.