Day 12

Part 1

Why not search for the Chief Historian near the gardener and his massive farm? There's plenty of food, so The Historians grab something to eat while they search.

You're about to settle near a complex arrangement of garden plots when some Elves ask if you can lend a hand. They'd like to set up fences around each region of garden plots, but they can't figure out how much fence they need to order or how much it will cost. They hand you a map (your puzzle input) of the garden plots.

Each garden plot grows only a single type of plant and is indicated by a single letter on your map. When multiple garden plots are growing the same type of plant and are touching (horizontally or vertically), they form a region. For example:

```
AAAA
BBCD
BBCC
EEEC
```

This 4x4 arrangement includes garden plots growing five different types of plants (labeled A, B, C, D, and E), each grouped into their own region.

In order to accurately calculate the cost of the fence around a single region, you need to know that region's area and perimeter.

The area of a region is simply the number of garden plots the region contains. The above map's type A, B, and C plants are each in a region of area 4. The type E plants are in a region of area 3; the type D plants are in a region of area 1.

Each garden plot is a square and so has four sides. The perimeter of a region is the number of sides of garden plots in the region that do not touch another garden plot in the same region. The type A and C plants are each in a region with perimeter 10. The type B and E plants are each in a region with perimeter 8. The lone D plot forms its own region with perimeter 4.

Visually indicating the sides of plots in each region that contribute to the perimeter using - and |, the above map's regions' perimeters are measured as follows:

```
+-+-+-+-+
|A A A A|
+-+-+-+-+     +-+
              |D|
+-+-+   +-+   +-+
|B B|   |C|
+   +   + +-+
|B B|   |C C|
+-+-+   +-+ +
          |C|
+-+-+-+   +-+
|E E E|
+-+-+-+
```

Plants of the same type can appear in multiple separate regions, and regions can even appear within other regions. For example:

```
00000
0X0X0
00000
0X0X0
00000
```

The above map contains five regions, one containing all of the O garden plots, and the other four each containing a single X plot.

The four X regions each have area 1 and perimeter 4. The region containing 21 type O plants is more complicated; in addition to its outer edge contributing a perimeter of 20, its boundary with each X region contributes an additional 4 to its perimeter, for a total perimeter of 36.

Due to "modern" business practices, the price of fence required for a region is found by multiplying that region's area by its perimeter. The total price of fencing all regions on a map is found by adding together the price of fence for every region on the map.

In the first example, region `A` has price `4 * 10 = 40`, region `B` has price `4 * 8 = 32`, region `C` has price `4 * 10 = 40`, region `D` has price `1 * 4 = 4`, and region `E` has price `3 * 8 = 24`. So, the total price for the first example is `140`.

In the second example, the region with all of the `O` plants has price `21 * 36 = 756`, and each of the four smaller `X` regions has price `1 * 4 = 4`, for a total price of `772` (756 + 4 + 4 + 4 + 4).

Here's a larger example:

```
RRRRIICCFF
RRRRIICCCF
VVRRRCCFFF
VVRCCCJFFF
VVVVCJJCFE
VVIVCCJJEE
VVIIICJJEE
MIIIIIJJEE
MIIISIJEEE
MMMISSJEEE
```

It contains:

- A region of R plants with price 12 * 18 = 216.
- A region of I plants with price 4 * 8 = 32.
- A region of C plants with price 14 * 28 = 392.
- A region of F plants with price 10 * 18 = 180.
- A region of V plants with price 13 * 20 = 260.
- A region of J plants with price 11 * 20 = 220.
- A region of C plants with price 1 * 4 = 4.
- A region of E plants with price 13 * 18 = 234.
- A region of I plants with price 14 * 22 = 308.
- A region of M plants with price 5 * 12 = 60.
- A region of S plants with price 3 * 8 = 24.

So, it has a total price of `1930`.

What is the total price of fencing all regions on your map?

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

import networkx as nx
```

In [2]:
```
test_path = Path("data/day12_test.txt")
data_path = Path("data/day12_data.txt")


def load_data(fpath: Path) -> nx.Graph:
    """Returns a Graph of field locations/crops."""
    cropmap = nx.Graph()

    cropmap.add_node((None, None), crop=None)  # Add NULL node to represent outside t

    with fpath.open() as ifh:
        rows = list(enumerate(ifh.readlines()))
        nrows = len(rows)
        ncols = len(rows[0][1]) - 1

        # Iterate over nodes, and add edges linking them labelled as "region" or "fen
        # This makes finding connected components easy later, as we can just remove t
        # "fence edges"
        for ridx, row in rows:
            for cidx, crop in enumerate(list(row.strip())):
                loc = (ridx, cidx)
                # Nodes are decorated with a set of fence directions (from NSEW), and
                # a count of corners.
                # We're using the corners to calculate the answer to part 2, and the
                # fence directions to help us calculate the number of corners
                cropmap.add_node(loc, crop=crop, fences=set(), corners=0)  # Add this

                # Add edges
                # Link N
                if ridx == 0:
                    cropmap.add_node((ridx-1, cidx), crop=None)  # Dummy node
                if cropmap.nodes[(ridx-1, cidx)]["crop"] == crop:  # Same crop
                    edgetype = "region"
                else:
                    edgetype = "fence"
                cropmap.add_edge(loc, (ridx-1, cidx), edgetype=edgetype)
                # Link W
                if cidx == 0:
                    cropmap.add_node((ridx, cidx-1), crop=None)  # Dummy node
                if cropmap.nodes[(ridx, cidx-1)]["crop"] == crop:  # Same crop
                    edgetype = "region"
                else:
                    edgetype = "fence"
                cropmap.add_edge(loc, (ridx, cidx-1), edgetype=edgetype)
                # Link S
                if ridx == nrows-1:
                    cropmap.add_node((ridx+1, cidx), crop=None)  # Dummy node
                    cropmap.add_edge(loc, (ridx+1, cidx), edgetype="fence")
                # Link E
                if cidx == ncols-1:
                    cropmap.add_node((ridx, cidx+1), crop=None)  # Dummy node
                    cropmap.add_edge(loc, (ridx, cidx+1), edgetype="fence")

                # Add fences to node "fences" attributes
                # N fence?
                if ridx == 0:
                    cropmap.nodes[loc]["fences"].add("N")
```

```
                elif cropmap.nodes[(ridx-1, cidx)]["crop"] != crop:  # not same crop
                    cropmap.nodes[loc]["fences"].add("N")
                    cropmap.nodes[(ridx-1, cidx)]["fences"].add("S")
                # W fence?
                if cidx == 0:  # Left col, so W fence
                    cropmap.nodes[loc]["fences"].add("W")
                elif cropmap.nodes[(ridx, cidx-1)]["crop"] != crop:  # not same crop
                    cropmap.nodes[loc]["fences"].add("W")
                    cropmap.nodes[(ridx, cidx-1)]["fences"].add("E")
                # E fence?
                if cidx == ncols - 1:  # Only at rightmost column
                    cropmap.nodes[loc]["fences"].add("E")
                # S fence?
                if ridx == nrows - 1:  # Only at rightmost column
                    cropmap.nodes[loc]["fences"].add("S")

    # Iterate over all nodes in the map and identify how many corners it
    # participates in. For clarity, in the example below:
    #
    # XXXX
    # XAXX
    # XAAX
    # XXXX
    #
    # The top A has two outer corners
    # The bottom left A has one outer corner and one inner corner
    # The bottom right A has two outer corners
    # For a total of six corners
    for ridx in range(nrows):
        for cidx in range(ncols):
            loc = (ridx, cidx)
            crop = cropmap.nodes[loc]["crop"]
            # Outer corners can be identified from corner fences meeting
            outcorners = sum([len(_.intersection(cropmap.nodes[loc]["fences"])) == 2
                              [{"N", "W"}, {"N", "E"}, {"S", "W"}, {"S", "E"}]])
            # Inner corners have to be identified by having neighbouring same crop in
            # but a different crop on the diagonal
            incorners = 0
            for (yc1, xc1, yc2, xc2, ydiag, xdiag) in [(-1, 0, 0, 1, -1, 1),
                                                       (0, 1, 1, 0, 1, 1),
                                                       (1, 0, 0, -1, 1, -1),
                                                       (0, -1, -1, 0, -1, -1)]:
                # Does this node have inner corners?
                if crop == cropmap.nodes[(ridx+yc1, cidx+xc1)]["crop"] == \
                           cropmap.nodes[(ridx+yc2, cidx+xc2)]["crop"] and \
                           crop != cropmap.nodes[(ridx+ydiag, cidx+xdiag)]["crop"]:
                    incorners += 1
            cropmap.nodes[loc]["corners"] = outcorners + incorners

    return cropmap, (nrows, ncols)


def find_regions(cropmap: nx.Graph) -> Iterable[nx.Graph]:
    """Returns a generator of connected components from the graph."""
    # Remove all edges that are fences
    unfenced = cropmap.copy(cropmap)
    fences = [(uloc, vloc) for (uloc, vloc, etype) in unfenced.edges(data="edgetype")
    unfenced.remove_edges_from(fences)

    # Remove all non-crop locations
    noncrop = [loc for (loc, crop) in unfenced.nodes(data="crop") if crop is None]
    unfenced.remove_nodes_from(noncrop)

    # Return connected components, which are single-crop connected regions
    return nx.connected_components(unfenced)
```

```
def cost_regions(cropmap: nx.Graph, regions: Iterable[nx.Graph]) -> list[dict]:
    result = []

    for component in list(regions):
        locs = sorted(list(component))  # Nodes in the region
        crop = cropmap.nodes[locs[0]]["crop"]  # Identify the region's crop
        fences = 0
        altfences = 0
        corners = 0
        for loc in locs:
            fences += len(cropmap.nodes[loc]["fences"])
            corners += cropmap.nodes[loc]["corners"]

        result.append({"crop": crop, "size": len(component),
                       "fences": fences, "fencecost": len(component) * fences,
                       "sides": corners, "sidecost": len(component) * corners})

    return result

cropmap, shape = load_data(test_path)
regions = find_regions(cropmap)
costs = cost_regions(cropmap, regions)
sum([_["fencecost"] for _ in costs])
```

Out[2]: 1930

In [3]:
```
cropmap, shape = load_data(data_path)
regions = find_regions(cropmap)
costs = cost_regions(cropmap, regions)
sum([_["fencecost"] for _ in costs])
```

Out[3]: 1461806

Part 2

Fortunately, the Elves are trying to order so much fence that they qualify for a bulk discount!

Under the bulk discount, instead of using the perimeter to calculate the price, you need to use the number of sides each region has. Each straight section of fence counts as a side, regardless of how long it is.

Consider this example again:

```
AAAA
BBCD
BBCC
EEEC
```

The region containing type A plants has 4 sides, as does each of the regions containing plants of type B, D, and E. However, the more complex region containing the plants of type C has 8 sides!

Using the new method of calculating the per-region price by multiplying the region's area by its number of sides, regions A through E have prices 16, 16, 32, 4, and 12, respectively, for a total price of 80.

The second example above (full of type X and O plants) would have a total price of 436.

Here's a map that includes an E-shaped region full of type E plants:

```
EEEEE
EXXXX
EEEEE
EXXXX
EEEEE
```

The E-shaped region has an area of 17 and 12 sides for a price of 204. Including the two regions full of type X plants, this map has a total price of 236.

This map has a total price of 368:

```
AAAAAA
AAABBA
AAABBA
ABBAAA
ABBAAA
AAAAAA
```

It includes two regions full of type B plants (each with 4 sides) and a single region full of type A plants (with 4 sides on the outside and 8 more sides on the inside, a total of 12 sides). Be especially careful when counting the fence around regions like the one full of type A plants; in particular, each section of fence has an in-side and an out-side, so the fence does not connect across the middle of the region (where the two B regions touch diagonally). (The Elves would have used the Möbius Fencing Company instead, but their contract terms were too one-sided.)

The larger example from before now has the following updated prices:

- A region of R plants with price 12 * 10 = 120.
- A region of I plants with price 4 * 4 = 16.
- A region of C plants with price 14 * 22 = 308.
- A region of F plants with price 10 * 12 = 120.
- A region of V plants with price 13 * 10 = 130.
- A region of J plants with price 11 * 12 = 132.
- A region of C plants with price 1 * 4 = 4.
- A region of E plants with price 13 * 8 = 104.
- A region of I plants with price 14 * 16 = 224.
- A region of M plants with price 5 * 6 = 30.
- A region of S plants with price 3 * 6 = 18.

Adding these together produces its new total price of `1206`.

What is the new total price of fencing all regions on your map?

In [4]:
```
# The insight here, implemented in the solution to part 1, is that
# the number of corners is the same as the number of sides. The
# number of corners is far easier to calculate than the number of
# sides.
# Reader, I can't tell you how long it took me to realise this.

cropmap, shape = load_data(test_path)
regions = find_regions(cropmap)
costs = cost_regions(cropmap, regions)
sum([_["sidecost"] for _ in costs])
```

Out[4]:
```
1206
```

In [5]:
```
cropmap, shape = load_data(data_path)
regions = find_regions(cropmap)
```

```
costs = cost_regions(cropmap, regions)
sum([_["sidecost"] for _ in costs])
```

Out[5]: 887932

In [ ]: