```
using PlutoUI
```

```
sort_str (generic function with 1 method)
```

```
function sort_str(str)
    return str |> collect |> sort |> join
end
```

```
parse_line (generic function with 1 method)
```

```
function parse_line(line::String)
    signal_raw, output_raw = split(line, " | ")
    return (signal=sort_str.(split(signal_raw)), output=sort_str.(split(output_raw)))
end
```

```
parse_file (generic function with 1 method)
```

```
function parse_file(io::IO)
    return [parse_line(line) for line in eachline(io)]
end
```

Problem 1

```
count_easy_digits (generic function with 1 method)
```

```
function count_easy_digits(data)
    # 1 => 2, 4 => 4, 7 => 3, 8 => 7
    unique_lengths = [2, 3, 4, 7]

    outputs = cat(getindex.(data, 2)..., dims=1)
    output_lengths = length.(outputs)
    return count(l -> l in unique_lengths, output_lengths)
end
```

```
349
```

```
  0.001191 seconds (1.82 k allocations: 384.172 KiB)
```

```
with_terminal() do
    open("./Day8/prob_input.txt") do io
        data = parse_file(io)
        @time count_easy_digits(data)
    end
end
```

Problem 2

As the signals part of the input will allways go over each digit `0..9` which means we can gradually figure out which characters correspond to which numbers. Given the following as default

```
  aaaa
 b    c
 b    c
  dddd
 e    f
 e    f
  gggg
```

Now lets saw when we see `length(signal) == 2` then we know its 1 and when its `length(signal) == 3` then its 7. So we can figure out a e.g.

```
ab: 1
dab: 7
```

So we know $d = a$. With the following signal input

```
acedgfb: 8
cdfbe: 5
gcdfa: 2
fbcad: 3
dab: 7
cefabd: 9
cdfgeb: 6
eafb: 4
cagedb: 0
ab: 1
```

So now if we sort 4 then it will be `aefb` and then check among signals if `length.(signal) == 5` that the delta between 4 and them should be only 1.

In above example if we see the we get (lets sort for easy interpretation)

```
bcdef: 5
abcdf: 3

-----

abef: 4
```

As we know representation of $1 = ab$ so among 3 and 5 we can figure out which is which and by process of skipping 2 we know that as well. So by now we have figured out $1, 2, 3, 4, 5, 7, 8$.

Now to distinguish between $0, 6, 9$ lets find a signal that does not have a 5 signal in it.

```
abcdef: 9
bcdefg: 6
abcdeg: 0

-----
bcdef: 5
```

So now we can see that $0 = abcdeg$. Now among those find all the ones which has chars of 4 and that number will be 9 and other will be 6.

```
abcdef: 9
bcdefg: 6

----

abef: 4
```

We have figured out all the numbers.

```
get_delta_count (generic function with 1 method)
```

```
function get_delta_count(l, r)
    return length(r) - sum([c ∈ l for c in r])
end
```

```
find_signal_and_output (generic function with 1 method)
```

```
function find_signal_and_output(data)
    (signal, output) = data
    length_to_signal = let
        signal_length = length.(signal)
        d = Dict()
        for (index, l) in enumerate(signal_length)
            val = get(d, l, [])
            push!(val, (index, signal[index]))
            d[l] = val
        end

        d
    end

    num1 = length_to_signal[2][1][2]
    num4 = length_to_signal[4][1][2]
    num_map = Dict(
        num1 => "1",
        num4 => "4",
        length_to_signal[3][1][2] => "7",
        length_to_signal[7][1][2] => "8")

    # Find amoung 2,3,5
    signal_of_length_5 = getindex.(length_to_signal[5],2)
    sl5_delta_num4 = get_delta_count.(signal_of_length_5, num4)

    num2 = signal_of_length_5[findall(l -> l == 2, sl5_delta_num4)[1]]
    num_map[num2] = "2"

    num_3_or_5 = signal_of_length_5[findall(l -> l == 1, sl5_delta_num4)]
    sl3or5_delta_num1 = get_delta_count.(num_3_or_5, num1)

    num5_index = findall(l -> l == 1, sl3or5_delta_num1)[1]
    num3_index = num5_index == 1 ? 2 : 1
    num3 = num_3_or_5[num3_index]
    num5 = num_3_or_5[num5_index]
    num_map[num3] = "3"
    num_map[num5] = "5"

    # Find among 0,6,9
    signal_of_length_6 = getindex.(length_to_signal[6],2)
    sl6_delta_num5 = get_delta_count.(signal_of_length_6, num5)
    num0 = signal_of_length_6[findall(l -> l == 1, sl6_delta_num5)[1]]
    num_map[num0] = "0"

    num_6_or_9 = signal_of_length_6[findall(l -> l == 0, sl6_delta_num5)]
    sl6or9_delta_num4 = get_delta_count.(num_6_or_9, num4)
    #return sl6or9_delta_num4, num_6_or_9, sl6_delta_num5, num5, signal_of_length_6

    num9_index = findall(l -> l == 0, sl6or9_delta_num4)[1]
    num6_index = num9_index == 1 ? 2 : 1
    num_map[num_6_or_9[num9_index]] = "9"
    num_map[num_6_or_9[num6_index]] = "6"
    return [num_map[o] for o in output] |> join |> s -> parse(Int64, s)
end
```

```
1070957

    0.006089 seconds (23.20 k allocations: 1.464 MiB)
```

```
with_terminal() do
```

After working on the above solution, I realized that it was very cumbersome. Though it works but its super complicated and does it by hand, at least the part of figuring out patterns for all numbers. I'm pretty sure there is more easier approach so I would have instead explored finding the position of each of a, b, c, d, e, f, g and then based on that figured out the number. I think that might have resulted in a simpler approach.