Advent of Code - Day 2 - Password Philosophy

Aaron Kaw

December 8, 2020

1 Part One

Your flight departs in a few days from the coastal airport; the easiest way down to the coast from here is via **toboggan**.

The shopkeeper at the North Pole Toboggan Rental Shop is having a bad day. "Something's wrong with our computers; we can't log in!" You ask if you can take a look.

Their password database seems to be a little corrupted: some of the passwords wouldn't have been allowed by the Official Toboggan Corporate Policy that was in effect when they were chosen. To try to debug the problem, they have created a list (your puzzle input) of **passwords** (according to the corrupted database) and **the corporate policy when that password was set**.

For example, suppose you have the following list:

```
1-3 a: abcde
1-3 b: cdefg
2-9 c: ccccccccc
```

Each line gives the password policy and then the password. The password policy indicates the lowest and highest number of times a given letter must appear for the password to be valid. For example, `1-3 a` means that a password must contain `a` at least `1` time and at most `3` times.

In the above example, `2` passwords are valid. The middle password, `cdefg`, is not; it contains no instances of `b`, but needs at least `1`. The first and third passwords are valid: they contain one `a` or nine `c`, both within the limits of their respective policies.

How many passwords are valid according to their policies?

```julia
struct PolicyAndPassword
    val1::Integer
    val2::Integer
    letter::Char
    password::String
end

function PolicyAndPassword(line::String)
    spl = split(line, " ")
    vals = split(spl[1], "-")
    PolicyAndPassword(
        tryparse(Int, vals[1]),
        tryparse(Int, vals[2]),
        spl[2][1],
        spl[end]
    )
end

pws = open("puzzle_input.txt") do file
    [PolicyAndPassword(line) for line ∈ eachline(file)]
end
```

1.1 Solution

```
parse_password(pw::PolicyAndPassword) = pw.val1 ≤ length(filter(l -> l == pw.letter,
pw.password)) ≤ pw.val2

are_valid = parse_password.(pws) |> sum
```

```
582
```

2 Part Two

While it appears you validated the passwords correctly, they don't seem to be what the Official Toboggan Corporate Authentication System is expecting.

The shopkeeper suddenly realizes that he just accidentally explained the password policy rules from his old job at the sled rental place down the street! The Official Toboggan Corporate Policy actually works a little differently.

Each policy actually describes two **positions in the password**, where `1` means the first character, `2` means the second character, and so on. (Be careful; Toboggan Corporate Policies have no concept of "index zero"!) **Exactly one of these positions** must contain the given letter. Other occurrences of the letter are irrelevant for the purposes of policy enforcement.

Given the same example list from above:

- `1-3 a: abcde` is **valid**: position `1` contains `a` and position `3` does not.
- `1-3 b: cdefg` is **invalid**: neither position `1` nor position `3` contains `b`.
- `2-9 c: ccccccccc` is **invalid**: both position `2` and position `9` contain `c`.

How many passwords are valid according to the new interpretation of the policies?

2.1 Solution

```
parse_password(pw::PolicyAndPassword) = xor(pw.password[pw.val1] == pw.letter,
pw.password[pw.val2] == pw.letter)

are_valid = parse_password.(pws) |> sum
```

```
729
```