Advent of Code 2020

Aaron Kaw

Day 5 - Binary Boarding

1 Part One

You board your plane only to discover a new problem: you dropped your boarding pass! You aren't sure which seat is yours, and all of the flight attendants are busy with the flood of people that suddenly made it through passport control.

You write a quick program to use your phone's camera to scan all of the nearby boarding passes (your puzzle input); perhaps you can find your seat through process of elimination.

Instead of **zones or groups**, this airline uses binary space partitioning to seat people. A seat might be specified like `FBFBBFFRLR`, where `F` means "front", `B` means "back", `L` means "left", and `R` means "right".

The first 7 characters will either be `F` or `B`; these specify exactly one of the **128 rows** on the plane (numbered `0` through `127`). Each letter tells you which half of a region the given seat is in. Start with the whole list of rows; the first letter indicates whether the seat is in **front** (`0` through `63`) or the **back** (`64` through `127`). The next letter indicates which half of that region the seat is in, and so on until you're left with exactly one row.

For example, consider just the first seven characters of `FBFBBFFRLR`:

- Start by considering the whole range, rows `0` through `127`.
- `F` means to take the **lower half**, keeping rows `0` through `63`.
- `B` means to take the **upper half**, keeping rows `32` through `63`.
- F means to take the lower half, keeping rows 32 through 47.
- B means to take the upper half, keeping rows 40 through 47.
- B keeps rows 44 through 47.
- F keeps rows 44 through 45.
- The final F keeps the lower of the two, row 44.

The last three characters will be either `L` or `R`; these specify exactly one of the **8 columns** of seats on the plane (numbered `0` through `7`). The same process as above proceeds again, this time with only three steps. `L` means to keep the **lower half** while `R` means to keep the **upper half**.

For example, consider just the last 3 characters of `FBFBBFFRLR`:

- Start by considering the whole range, columns 0 through 7.
- R means to take the upper half, keeping columns 4 through 7.
- L means to take the lower half, keeping columns 4 through 5.
- The final R keeps the upper of the two, column 5.

So, decoding `FBFBBFFRLR` reveals that it is the seat at **row 44, column 5**.

Every seat also has a unique **seat ID**: multiply the row by 8, then add the column. In this example, the seat has ID `44 * 8 + 5 = 357`.

Here are some other boarding passes:

- BFFFBBFRRR: row 70, column 7, seat ID 567.
- FFFBBBFRRR: row 14, column 7, seat ID 119.
- BBFFBBFRLL: row 102, column 4, seat ID 820.

```julia
function parse_!(ch::Char, vals::AbstractVector{Int}, part::NTuple{2, Char})
    if ch == part[1]
        N = vals |> length |> Int
        deleteat!(vals, N÷2+1:N)
    elseif ch == part[2]
        N = vals |> length |> Int
        deleteat!(vals, 1:N÷2)
    else
        ErrorException("Invalid binary space partition indicator.") |> throw
    end
end
parse_row!(ch, bsp_row) = parse_!(ch, bsp_row, ('F', 'B'))
parse_col!(ch, bsp_col) = parse_!(ch, bsp_col, ('L', 'R'))

struct Seat
    bsp::String
    row::Int
    col::Int
    id::Int
end

function Seat(bsp::String)
    bsp_row = bsp[1:7]
    bsp_col = bsp[8:10]

    row = 0:127 |> collect
    for ch ∈ bsp_row
        parse_row!(ch, row)
    end
    col = 0:7 |> collect
    for ch ∈ bsp_col
        parse_col!(ch, col)
    end

    r = row[1]
    c = col[1]
    return Seat(bsp, r, c, 8r + c)
```

```
end

Base.show(io::IO, seat::Seat) = print(io, "(", seat.row, ", ", seat.col, ", ", seat.id,
")")

@show Seat("FBFBBFFRLR");
@show Seat("BFFFBBFRRR");
@show Seat("FFFBBBFRRR");
@show Seat("BBFFBBFRLL");
```

```
Seat("FBFBBFFRLR") = (44, 5, 357)
Seat("BFFFBBFRRR") = (70, 7, 567)
Seat("FFFBBBFRRR") = (14, 7, 119)
Seat("BBFFBBFRLL") = (102, 4, 820)
```

```
seats = open("puzzle_input.txt") do file
    seats = Vector{Seat}(undef, 0)
    for line ∈ eachline(file)
        seat = Seat(line)
        push!(seats, seat)
    end
    return seats
end
```

```
901-element Array{Main.##WeaveSandBox#299.Seat,1}:
 (111, 6, 894)
 (33, 2, 266)
 (44, 5, 357)
 (34, 5, 277)
 (24, 3, 195)
 (28, 7, 231)
 (90, 2, 722)
 (68, 3, 547)
 (57, 1, 457)
 (102, 6, 822)

 ⋮@*((20, 5, 165)(107, 0, 856)(4, 2, 34)(54, 0, 432)(86, 2, 690)(6, 1, 49)(91, 6,
734)(57, 6, 462)(60, 4, 484)
```

As a sanity check, look through your list of boarding passes. **What is the highest seat ID on a boarding pass?**

```
max_id = [seat.id for seat ∈ seats] |> maximum
```

```
908
```

2 Part Two

Ding! The "fasten seatbelt" signs have turned on. Time to find your seat.

It's a completely full flight, so your seat should be the only missing boarding pass in your list. However, there's a catch: some of the seats at the very front and back of the plane don't exist on this aircraft, so they'll be missing from your list as well.

Your seat wasn't at the very front or back, though; the seats with IDs +1 and -1 from yours will be in your list.

What is the ID of your seat?

```julia
ids = [seat.id for seat ∈ seats] |> sort
the_diff, my_idx = findmax(ids |> diff)
my_id = ids[my_idx] + 1
```

```
619
```