day5

December 6, 2020

0.1 Day 5 - Boarding Passes

This is an interesting one. There's a lot of words that essentially describe a binary partition tree. I know that this will be easiest if I use a data structure or something to parse that tree and identify the seat, I just know I'm going to need this in part 2.

But, finding the highest seat id in the dataset doesn't actually require us to understand or parse the tree at all. Instead we can note that all we need to do is sort the list of seat ids.

This will work because we are guaranteed due to the BSP's, certain things. `Fx no matter the x will always be higher than Bx` This means that we can say that FF > FB to find the row. Since the row number is multipled by 8, and there are 8 chairs in any row, we know that the highest seat id will be the largest rowid, and then the largest chairid within that row. Once we've done that, we can convert the words into numbers.

I've got some thoughts on the converting rows into numbers. I have ahunch that the way it's structured, FFFB is very similar to 1110 as a binary code, and that FFBF would be the same as 1101. To prove that, let's think about a plane with just 4 rows. From rear, 3 down to front 1, the codes would be `BB BF FB FF`

That feels like a good example so lets test it with the samples given, using either FB and using LR with the same code

[1]:
```
import pytest
import ipytest
ipytest.autoconfig()
```

[2]:
```
import bitstring

def parseseat(seat):
    a = bitstring.BitArray()
    for c in seat[:7]:
        if c == 'B':
            a += '0b1'
        else:
            a += '0b0'
    return a.uint

assert 44 == parseseat('FBFBBFFRLR')
assert 70 == parseseat('BFFFBBFRRR')
assert 14 == parseseat('FFFBBBFRRR')
assert 102 == parseseat('BBFFBBFRLL')
```

Great, but we want the column id as well, not just the row ID, so lets handle that the same way.

We can also note that timesing the first number by 8 is the same as shifting left 3, so this really is just the number in binary, so we can work that out at the same time.

[3]:
```
def parseseat(seat):
    a = bitstring.BitArray()
    b = bitstring.BitArray()
    for c in seat[:7]:
        if c == 'B':
            a += '0b1'
        else:
            a += '0b0'
    for c in seat[7:]:
        if c == 'R':
            b += '0b1'
        else:
            b += '0b0'

    return a.uint,b.uint,(a+b).uint

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

So now we know that we can turn them into a number easily, we can work out if we want to sort them by parsing them and turning them into numbers, or whether we can just sort them as strings. It feels easiest to find the highest by simply going through the list, turning it into a number, and then storing the highest we find…

[4]:
```
maxseatid = 0
for line in open('day5.txt').readlines():
    line = line.strip() # Get rid of those newlines
    _,_,id = parseseat(line)
    maxseatid = max(id, maxseatid)
print(maxseatid)
```

```
806
```

0.2 Part 2 - Find the missing seat

This should be fairly easy, get the seat ids, sort them, and then iterate through them looking to see if it is not prev+1, if not, we've found it.

[5]:

```
ids = sorted([parseseat(line.strip())[2] for line in open('day5.txt').
 ↪readlines()])
lastid = ids[0]
for id in ids[1:]:
    if id > lastid+1:
        print(lastid+1)
    lastid = id
```

```
562
```