day13

December 15, 2020

1 Reading timetables

This is one of those where I think, it can't possibly be this easy.

We have a set of busses, each is on a looping timetable that loops every n minutes. The question is at minute x, which is the next bus to arrive and how long until it leaves.

If we assume each bus is simply modular arithmatic (or clock arithmatic), then all we need to do is iterate over all the buses for the given time and see what the lowest value is. Whatever that is, is the next bus to arrive

[6]:
```
import ipytest
ipytest.autoconfig()

bus = [7,13,59,31,19]

def bus_at_time(buslist, time):
    return [time % bus for bus in buslist]

assert bus_at_time(bus, 939) == [1, 3, 54, 9, 8]
```

Except that's not right, because the answer we want is the third bus….

We need to work out distance to next stop, which is simply bus number - time % bus

[8]:
```
def bus_at_time(buslist, time):
    return [bus - (time % bus) for bus in buslist]

assert bus_at_time(bus, 939) == [6, 10, 5, 22, 11]
```

Great, solving for the lowest time to distance gets us the right answer, now all we need to do is multiply it by the original bus number. That's probably easier if we zip the answers together to make tuples before finding the minimum

[13]:
```
def find_next_bus(buslist, time):
    nextdue = bus_at_time(buslist, time)
    pairs = zip(nextdue,buslist)
    return min(pairs)

t,i = find_next_bus(bus, 939)
```

```
assert t*i == 295
```

Now to try the same on the real data

[19]:
```
f = open("day13.txt").readlines()
time = int(f[0].strip())
bustable = [int(b) for b in f[1].split(",") if b != "x"]
t,i = find_next_bus(bustable, time)
assert t*i == 2935
```

1.1 Part 2 - Like orbiting planets

Part 1 was simpler than I thought. Part 2 means working out something more mathmatically complicated. Those who did last years will remember a puzzle where one had to find the point at which orbiting planets line up, which was a very similar problem, since our buses are orbiting the station and coming back every n minutes as well.

If we think of them as orbits, we only need to think of when the largest one is in the right place, before worrying about the others. We can then filter down the list of time stamps to just those that match when the largest is in the right place, and look to see those where the next largest is in teh right place.

However, even thinking about this made me go and have a lie down on how to implement it, so I stopped and waited for day 14 instead