# day8

December 12, 2020

## 0.1 Day 8 - Is this revenge of IntCode?

My first look at this and I'm beginning to dread that we'll have something similar to IntCode from last year. I hated my IntCode computer by half way through and it was one of the reasons for dropping out last year, as I spent more time debugging my virtual computer than I did trying to solve the problem.

Anyway, let's not do that this time.

This part 1 can almost certainly be solved by fairly simple means, let's not overcomplicate things. We load the lines into an list of strings, have a line number, and start at 1. For each string we parse the string, either add to the accumulator, or change the line number, add 1 to the line number and then go again. But we'll add a set of seen lines as we go over them, and check whether we've already seen the line, if we have, print out the value of the accumulator and quit.

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

lines = [
    "nop +0",
"acc +1",
"jmp +4",
"acc +3",
"jmp -3",
"acc -99",
"acc +1",
"jmp -4",
"acc +6"
]

def parse1(lines):
    line = 0
    seen = set()
    acc = 0
    while True:
        currentline = lines[line]
        seen.add(line)
        if currentline[:3] == "jmp":
            line += int(currentline[4:])
```

```
        else:
            if currentline[:3] == "acc":
                acc += int(currentline[4:])
            # Always increment line after acc or nop
            line += 1
        if line in seen:
            break
    return acc

assert parse1(lines) == 5
```

Great, that works, let's try that on the real data

[2]:
```
lines = [line.strip() for line in open('day8.txt')]
print(parse1(lines))
```

```
1548
```

### 0.2 Part 2 - Mutate the program

This looks horrible at first sight, and it actually is. What we need to do is mutate each jmp line into a nop line, and rerun the program, keep doing this until we find one that ends properly.

In this case, there are now 2 ending conditions, we ran off the end of the program (success) and we hit a loop (fail), so we'll need to return not just the accumulator value, but which one we did.

We then need to iterate through each line, swapping a nop for a jmp and a jmp for a nop, and try the program, until we get to the end.

[3]:
```
import ipytest
ipytest.autoconfig()

lines_loop = [
    "nop +0",
"acc +1",
"jmp +4",
"acc +3",
"jmp -3",
"acc -99",
"acc +1",
"jmp -4",
"acc +6"
]

lines_end = [
    "nop +0",
"acc +1",
"jmp +4",
"acc +3",
```

```
"jmp -3",
"acc -99",
"acc +1",
"nop -4",
"acc +6"
]

def parse2(lines):
    line = 0
    seen = set()
    acc = 0
    while True:
        currentline = lines[line]
        seen.add(line)
        if currentline[:3] == "jmp":
            line += int(currentline[4:])
        else:
            if currentline[:3] == "acc":
                acc += int(currentline[4:])
            # Always increment line after acc or nop
            line += 1
        if line in seen:
            return (False,acc)
        if line >= len(lines):
            return (True,acc)

assert parse2(lines_loop) == (False,5)
assert parse2(lines_end) == (True,8)
```

Now let's try mutating all the lines on the sample

[4]:
```
def mutate_lines(lines):
    for i,line in enumerate(lines):
        newlist = lines.copy()
        if line[:3] == "nop":
            newlist[i] = line.replace("nop", "jmp")
        elif line[:3] == "jmp":
            newlist[i] = line.replace("jmp", "nop")
        ret,acc = parse2(newlist)
        if ret:
            return (True,acc)
    return (False,acc)

assert mutate_lines(lines_loop) == (True,8)
```

Great, that works just fine. Let's try it on the real data

[5]:
```
print(mutate_lines(lines))
```

```
(True, 1375)
```