Advent of Code

December 3, 2021

1 Advent of Code

```
[ ]: # set up the environment
     import numpy as np
```

1.1 Day 1: Sonar Sweep

You're minding your own business on a ship at sea when the overboard alarm goes off! You rush to see if you can help. Apparently, one of the Elves tripped and accidentally sent the sleigh keys flying into the ocean!

Before you know it, you're inside a submarine the Elves keep ready for situations like this. It's covered in Christmas lights (because of course it is), and it even has an experimental antenna that should be able to track the keys if you can boost its signal strength high enough; there's a little meter that indicates the antenna's signal strength by displaying 0-50 stars.

```
[ ]: # load the data from the source file 'data/depth.dat' and store it in an array.
     depths = np.loadtxt("data/depths.dat", dtype="int", delimiter=",", unpack=False)
     print(depths)
     print(len(depths))
```

```
[ 173  175  171 … 7118 7115 7121]
2000
```

1.1.1 Problem 1

As the submarine drops below the surface of the ocean, it automatically performs a sonar sweep of the nearby sea floor. On a small screen, the sonar sweep report (your puzzle input) appears: each line is a measurement of the sea floor depth as the sweep looks further and further away from the submarine.

```
[ ]: depth_increase_count = 0
     previous_depth = depths[0]
     for x in depths:
         if x > previous_depth:
             depth_increase_count += 1
         previous_depth = x

     print("Number of depth increase: ", depth_increase_count)
```

```
Number of depth increase:  1521
```

1.1.2 Part 2

Considering every single measurement isn't as useful as you expected: there's just too much noise in the data.

Instead, consider sums of a three-measurement sliding window.

[ ]:
```python
# create a new array of the sliding data
three_measurement_sums = []
for x in range(len(depths)-2):
    three_measurement_sums.append(depths[x]+depths[x+1]+depths[x+2])

# repeat the calculation with the above data
depth_increase_count = 0
previous_depth = three_measurement_sums[0]
for x in three_measurement_sums:
    if x > previous_depth:
        depth_increase_count += 1
    previous_depth = x

print("Number of depth increase: ", depth_increase_count)
```

```
Number of depth increase:  1543
```

1.2 Day 2: Dive!

1.2.1 Part One

It seems like the submarine can take a series of commands like forward 1, down 2, or up 3:

```
forward X increases the horizontal position by X units.
down X increases the depth by X units.
up X decreases the depth by X units.
```

Note that since you're on a submarine, down and up affect your depth, and so they have the opposite result of what you might expect.

The submarine seems to already have a planned course (your puzzle input). You should probably figure out where it's going. For example:

forward 5 down 5 forward 8 up 3 down 8 forward 2

Your horizontal position and depth both start at 0. The steps above would then modify them as follows:

```
forward 5 adds 5 to your horizontal position, a total of 5.
down 5 adds 5 to your depth, resulting in a value of 5.
forward 8 adds 8 to your horizontal position, a total of 13.
up 3 decreases your depth by 3, resulting in a value of 2.
down 8 adds 8 to your depth, resulting in a value of 10.
forward 2 adds 2 to your horizontal position, a total of 15.
```

After following these instructions, you would have a horizontal position of 15 and a depth of 10. (Multiplying these together produces 150.)

Calculate the horizontal position and depth you would have after following the planned course. What do you get if you multiply your final horizontal position by your final depth?

```
[ ]: # Import the manoeuvreing data into an array fo tuples
     manoeuvres = np.genfromtxt("data/manoeuvre.dat", dtype="U7, i4", delimiter=" ")
     print(manoeuvres[:10], '...', manoeuvres[-10:])
     print(len(manoeuvres))
```

```
[('forward', 8) ('forward', 9) ('forward', 9) ('down', 3) ('forward', 9)
 ('down', 1) ('down', 7) ('down', 7) ('down', 4) ('down', 2)] … [('down', 8)
('forward', 6) ('forward', 7) ('forward', 9) ('forward', 4)
 ('down', 3) ('up', 5) ('down', 7) ('down', 7) ('forward', 9)]
1000
```

```
[ ]: horizontal_pos = 0
     depth = 0
     for x in manoeuvres:
         direction = x[0]
         steps = x[1]
         if direction == "forward":
             horizontal_pos += steps
         if direction == "up":
             depth -= steps
         if direction == "down":
             depth += steps

     print("Displacement: ", horizontal_pos, ", Depth: ", depth)
     print("Product of displacement and dept: ", horizontal_pos * depth)
```

```
Displacement:  2003 , Depth:  872
Product of displacement and dept:  1746616
```

1.2.2 Part Two

Based on your calculations, the planned course doesn't seem to make any sense. You find the submarine manual and discover that the process is actually slightly more complicated.

In addition to horizontal position and depth, you'll also need to track a third value, aim, which also starts at 0. The commands also mean something entirely different than you first thought:

```
down X increases your aim by X units.
up X decreases your aim by X units.
forward X does two things:
    It increases your horizontal position by X units.
    It increases your depth by your aim multiplied by X.
```

Again note that since you're on a submarine, down and up do the opposite of what you might expect: "down" means aiming in the positive direction.

Now, the above example does something different:

```
forward 5 adds 5 to your horizontal position, a total of 5.
    Because your aim is 0, your depth does not change.
down 5 adds 5 to your aim, resulting in a value of 5.
forward 8 adds 8 to your horizontal position, a total of 13.
    Because your aim is 5, your depth increases by 8*5=40.
up 3 decreases your aim by 3, resulting in a value of 2.
down 8 adds 8 to your aim, resulting in a value of 10.
forward 2 adds 2 to your horizontal position, a total of 15.
    Because your aim is 10, your depth increases by 2*10=20 to a total of 60.
```

After following these new instructions, you would have a horizontal position of 15 and a depth of 60. (Multiplying these produces 900.)

Using this new interpretation of the commands, calculate the horizontal position and depth you would have after following the planned course.

[ ]:
```
horizontal_pos = 0
depth = 0
aim = 0
for x in manoeuvres:
    direction = x[0]
    steps = x[1]
    if direction == "up":
        aim -= steps
    if direction == "down":
        aim += steps
    if direction == "forward":
        horizontal_pos += steps
        depth += aim * steps


print("Displacement: ", horizontal_pos, ", Depth: ", depth)
print("Product of displacement and dept: ", horizontal_pos * depth)
```

```
Displacement:  2003 , Depth:  869681
Product of displacement and dept:  1741971043
```

1.3 Day 3: Binary Diagnostic

1.3.1 Part 1

The submarine has been making some odd creaking noises, so you ask it to produce a diagnostic report just in case.

The diagnostic report (your puzzle input) consists of a list of binary numbers which, when decoded properly, can tell you many useful things about the conditions of the submarine. The first parameter to check is the power consumption.

You need to use the binary numbers in the diagnostic report to generate two new binary numbers (called the gamma rate and the epsilon rate). The power consumption can then be found by multiplying the gamma rate by the epsilon rate.

Each bit in the gamma rate can be determined by finding the most common bit in the corresponding position of all numbers in the diagnostic report. For example, given the following diagnostic report:

```
00100
11110
10110
10111
10101
01111
00111
11100
10000
11001
00010
01010
```

Considering only the first bit of each number, there are five 0 bits and seven 1 bits. Since the most common bit is 1, the first bit of the gamma rate is 1.

The most common second bit of the numbers in the diagnostic report is 0, so the second bit of the gamma rate is 0.

The most common value of the third, fourth, and fifth bits are 1, 1, and 0, respectively, and so the final three bits of the gamma rate are 110.

So, the gamma rate is the binary number 10110, or 22 in decimal.

The epsilon rate is calculated in a similar way; rather than use the most common bit, the least common bit from each position is used. So, the epsilon rate is 01001, or 9 in decimal. Multiplying the gamma rate (22) by the epsilon rate (9) produces the power consumption, 198.

Use the binary numbers in your diagnostic report to calculate the gamma rate and epsilon rate, then multiply them together. What is the power consumption of the submarine? (Be sure to represent your answer in decimal, not binary.)

[ ]:
```python
# Import the data into a text array - easier for getting positional data
diagnostics = np.genfromtxt("data/diagnostic.dat", dtype="U")
print(diagnostics[:10])
print(len(diagnostics))
```

```
['111010101100' '100001001100' '000111101100' '100100000000'
 '001001001110' '100110101011' '001001100101' '010000010110'
 '011011001001' '001001000101']
1000
```

[ ]:
```python
# get the number of bits in the data
bits = len(diagnostics[0])
print(bits)
```

```
12
```

[ ]:
```python
epsilon_str = ""
gamma_str = ""

for x in range(len(diagnostics[0])):
    gamma_val = 0
    for y in diagnostics:
        if y[x] == '1':
            gamma_val += 1
        else:
            gamma_val -= 1

# take advantage of the fact that the epsilon value = not(gamma value)
# (could be done later on the whole string but just as easy to build it now)
    if gamma_val > 0:
        gamma_str += '1'
        epsilon_str +=  '0'
    else:
        gamma_str += '0'
        epsilon_str += '1'

# convert the base 2 strings to integers
gamma_rate = int(gamma_str, 2)
epsilon_rate = int(epsilon_str, 2)

power_consumptions = gamma_rate * epsilon_rate

print("Gama rate: ", gamma_rate)
print("Epsilon rate: ", epsilon_rate)
print("Power Consumption: ", power_consumptions)
```

```
Gama rate:  3903
Epsilon rate:  192
Power Consumption:  749376
```

1.3.2 Part Two

Next, you should verify the life support rating, which can be determined by multiplying the oxygen generator rating by the CO2 scrubber rating.

Both the oxygen generator rating and the CO2 scrubber rating are values that can be found in your diagnostic report - finding them is the tricky part. Both values are located using a similar process that involves filtering out values until only one remains. Before searching for either rating value, start with the full list of binary numbers from your diagnostic report and consider just the first bit of those numbers. Then:

```
Keep only numbers selected by the bit criteria for the type of rating value for which you are s
    Discard numbers which do not match the bit criteria.
If you only have one number left, stop; this is the rating value for which you are searching.
Otherwise, repeat the process, considering the next bit to the right.
```

The bit criteria depends on which type of rating value you want to find:

To find oxygen generator rating, determine the most common value (0 or 1) in the current bit position, and keep only numbers with that bit in that position. If 0 and 1 are equally common, keep values with a 1 in the position being considered. To find CO2 scrubber rating, determine the least common value (0 or 1) in the current bit position, and keep only numbers with that bit in that position. If 0 and 1 are equally common, keep values with a 0 in the position being considered.

For example, to determine the oxygen generator rating value using the same example diagnostic report from above:

```
Start with all 12 numbers and consider only the first bit of each number.
    There are more 1 bits (7) than 0 bits (5)
    keep only the 7 numbers with a 1 in the first position:
        11110, 10110, 10111, 10101, 11100, 10000, and 11001.
Then, consider the second bit of the 7 remaining numbers:
    there are more 0 bits (4) than 1 bits (3)
    keep only the 4 numbers with a 0 in the second position:
        10110, 10111, 10101, and 10000.
In the third position,
    three of the four numbers have a 1
    keep those three:
        10110, 10111, and 10101.
In the fourth position,
    two of the three numbers have a 1
    keep those two:
        10110 and 10111.
In the fifth position,
    there are an equal number of 0 bits and 1 bits (one each).
    to find the oxygen generator rating, keep the number with a 1 in that position:
        10111.
As there is only one number left, stop;
    the oxygen generator rating is 10111, or 23 in decimal.
```

Then, to determine the CO2 scrubber rating value from the same example above:

```
Start again with all 12 numbers and consider only the first bit of each number.
    There are fewer 0 bits (5) than 1 bits (7)
    keep only the 5 numbers with a 0 in the first position:
        00100, 01111, 00111, 00010, and 01010.
Then, consider the second bit of the 5 remaining numbers:
    there are fewer 1 bits (2) than 0 bits (3)
    keep only the 2 numbers with a 1 in the second position:
        01111 and 01010.
In the third position, there are an equal number of 0 bits and 1 bits (one each).
    to find the CO2 scrubber rating, keep the number with a 0 in that position:
        01010.
As there is only one number left, stop;
    the CO2 scrubber rating is 01010, or 10 in decimal.
```

Finally, to find the life support rating, multiply the oxygen generator rating (23) by the CO2 scrubber rating (10) to get 230.

Use the binary numbers in your diagnostic report to calculate the oxygen generator rating and CO2 scrubber rating, then multiply them together. What is the life support rating of the submarine?

[ ]:

```
```