Advent of Code 2021, Day 24

1 Problem Statement

The Arithmetic Logic Unit (ALU) has four integer registers `w`, `x`, `y`, and `z`. At the start of the program, all registers have the value `0`. The ALU supports six instructions:

1. `inp a` – Read an input value and write it to `a`
2. `mul a b` – Multiply the value of `a` by `b`, then store the result in `a`
3. `div a b` – Divide the value of `a` by `b`, truncate the result to an integer (round towards 0), then store the result in `a`
4. `mod a b` – Divide the values of `a` by the value of `b`, then store the remainder in `a`
5. `eql a b` – If the value of `a` and `b` are equal, store `1` in `a`. Otherwise, store `0` in `a`

In each instruction, `a` will always be one of the four registers, and `b` can either be another register or an integer literal.

You are given an ALU program that has fourteen `inp` instructions corresponding to the digits of a *model number*. If the model number is valid, the program will exit with `z=0`. Any other value in `z` indicates an invalid number.

In the two parts of the puzzle, you must find the largest and smallest model numbers that are valid and do not contain any `0` digits.

2 Walkthrough

2.1 Basic Analysis

To begin, notice that the input program is composed of 252 lines, which can be divided into 14 discrete "blocks" of 18 lines each. Every block begins with an `inp` instruction to process the next digit of the model number, and is always structured as follows:

```
inp w
mul x 0
add x z
mod x 26
div z A
add x B
eql x w
eql x 0
mul y 0
add y 25
mul y x
add y 1
mul z y
mul y 0
add y w
add y C
mul y x
add z y
```

where `A`, `B`, and `C` are integer literals. Here are their values for each block (from my puzzle input):

Block	A	B	C
1	1	12	6
2	1	10	2
3	1	10	13
4	26	-6	8
5	1	11	13
6	26	-12	8
7	1	11	3
8	1	12	11
9	1	12	10
10	26	-2	8
11	26	-5	14
12	26	-4	6
13	26	-4	8
14	26	-12	2

Notice that `A` is always either `1` or `26`, and assumes no other values. Consider the pseudocode equivalent of a block.

```
w = input()
x *= 0          # x = 0
x += z          # x = z
x %= 26         # x = z % 26
z //= A
x += B          # x = z % 26 + B
x = (x == w)    # x = (z % 26 + B) == w
x = (x == 0)    # x = (z % 26 + B) != w
y *= 0          # y = 0
y += 25         # y = 25
y *= x          # y = 25 * x
y += 1          # y = 25 * x + 1
z *= y
y *= 0          # y = 0
y += w          # y = w
y += C          # y = w + C
y *= x          # y = (w + C) * x
z += y
```

The comments illustrate how this can be simplified to the following:

```
w = input()
x = (z % 26 + B) != w
y = 25 * x + 1
z //= A
z *= y
y = (w + C) * x
z += y
```

2.2 Additional Analysis

The simplified block from above can be rewritten as

```
1 w = input()
2 mult = 1 if ((z % 26 + B) == w) else 26
3 incr = 0 if ((z % 26 + B) == w) else w + C
4 z //= A  # A == 1 | A == 26
5 z *= mult
6 z += incr
```

Consider the value of `incr`. First, `w`, being a digit of the model number, must satisfy `1<=w<=9`. Additionally from observation of the puzzle input, `C` is always positive. Thus, on line 6, `z` is either left unchanged or incremented by a positive integer. Since the only other assignments to `z` are `mul` and `div` instructions with `1` or `26`, then `z` must necessarily be **non-negative** throughout the entire program's execution.

To gain further insight into the structure of this problem, let us now consider the blocks with `A == 1` (Type 1) and `A == 26` (Type 2) separately.

2.2.1 Type 1 Blocks

By observing the puzzle input, we have that `B > 9` for every Type 1 block. The consequences of this condition can be seen by examining the simplified pseudocode above – specifically, the condition

```
(z % 26 + B) == w
```

Since we know that `z` is guaranteed to be non-negative, `z % 26` must also be non-negative. Therefore for all Type 1 blocks, `(z % 26) + B > 0 + 9 = 9`. Since `w` is a digit between 1 and 9, then the condition must *always* be false, and thus Type 1 blocks can be further simplified to

```
w = input()
# z //= 1
z *= 26
z += w + C
```

We can now see that *every* Type 1 block will increase the value of `z`. In particular, `z` is first multiplied by 26, then incremented by a positive integer. Specifically, since `1<=w<=9` and (from observation of the input) `0<=C<=16`, the increment lies between 1 and 25, inclusive. As a consequence, it will be helpful to conceptualize these operations in base 26. In this context, each Type 1 block shifts the base-26 digits of `z` left once, then adds a non-zero base-26 digit.

2.2.2 Type 2 Blocks

Unfortunately, not much simplification can be done in this case. However, note that there is an *equal number* of Type 1 and Type 2 blocks in the program. Because every Type 1 block left-shifts and increments `z`, the only possible way for `z` to be `0` when the program terminates is for this enlarging effect to be "cancelled out" by a corresponding Type 2 block.

In particular, this "cancelling out" is accomplished by the `div z 26` instruction present in each Type 2 block, which effectively right-shifts the base-26 representation of `z` by one digit. Critically, however, if `(z % 26 + B) != w`, then `z` is subsequently *multiplied* by 26 again, nullifying the cancellation. Thus `z % 26 + B` must equal `w` for **every single** Type 2 block for the model number to be valid.

2.3 Constraint Generation

As mentioned, it is extremely helpful to think of `z` as a base-26 number. In this context, every Type 1 block left-shifts `z`, then adds `w + C` as the new least-significant digit. Type 2 blocks (assuming `(z % 26 + B) == w`) then right-shift `z`, cancelling out the growth from a Type 1 block. More aptly, `z` can be thought of as a base-26 stack, with Type 1 blocks pushing a base-26 digit onto the stack (as the least-significant digit), and Type 2 blocks (assuming the equality condition holds) popping the least-significant digit off of the stack.

What follows is a demonstration of stepping through the blocks from my puzzle input to obtain the constraints on each digit of the model number. The base-26 representation of `z` will be given as a list of integers between 0 and 25, each corresponding to a single digit of `z`. The fourteen digits of the model number are notated as d_1 through d_{14}.

```
z = 0
```

1. `Type 1, B=12, C=6`

```
z <<= 1
z += w+6
z = [d1+6]
```

2. `Type 1, B=10, C=2`

```
z <<= 1
z += w+2
z = [d1+6, d2+2]
```

3. `Type 1, B=10, C=13`

```
z <<= 1
z += w+13
z = [d1+6, d2+2, d3+13]
```

4. `Type 2, B=-6, C=8`

   Condition: `(z % 26 - 6) == w`

$$\begin{aligned}(d_3 + 13) - 6 &= d_4\\ d_3 + 7 &= d_4\end{aligned}$$

```
z >>= 1
z = [d1+6, d2+2]
```

5. `Type 1, B=11, C=13`

```
z <<= 1
z += w+13
z = [d1+6, d2+2, d5+13]
```

6. `Type 2, B=-12, C=8`

   Condition: `(z % 26 - 12) == w`

$$\begin{aligned}(d_5 + 13) - 12 &= d_6\\ d_5 + 1 &= d_6\end{aligned}$$

```
z >>= 1
z = [d1+6, d2+2]
```

7. `Type 1, B=11, C=3`

```
z <<= 1
z += w+3
z = [d1+6, d2+2, d7+3]
```

8. `Type 1, B=12, C=11`

```
z <<= 1
z += w+11
z = [d1+6, d2+2, d7+3, d8+11]
```

9. `Type 1, B=12, C=10`

```
z <<= 1
z += w+10
z = [d1+6, d2+2, d7+3, d8+11, d9+10]
```

10. `Type 2, B=-2, C=8`

    Condition: `(z % 26 - 2) == w`

$$\begin{aligned}(d_9 + 10) - 2 &= d_{10}\\ d_9 + 8 &= d_{10}\end{aligned}$$

```
z >>= 1
z = [d1+6, d2+2, d7+3, d8+11]
```

11. `Type 2, B=-5, C=14`

    Condition: `(z % 26 - 5) == w`

$$\begin{aligned}(d_8 + 11) - 5 &= d_{11}\\ d_8 + 6 &= d_{11}\end{aligned}$$

12. `Type 2, B=-4, C=6`

```
z >>= 1
z = [d1+6, d2+2, d7+3]
```

Condition: `(z % 26 - 4) == w`

$$\begin{aligned}(d_7 + 3) - 4 &= d_{12}\\ d_7 - 1 &= d_{12}\end{aligned}$$

13. `Type 2, B=-4, C=8`

```
z >>= 1
z = [d1+6, d2+2]
```

Condition: `(z % 26 - 4) == w`

$$\begin{aligned}(d_2 + 2) - 4 &= d_{13} \\ d_2 - 2 &= d_{13}\end{aligned}$$

```
z >>= 1
z = [d1+6]
```

14. `Type 2, B=-12, C=2`

    Condition: `(z % 26 - 12) == w`

$$\begin{aligned}(d_1 + 6) - 12 &= d_{14} \\ d_1 - 6 &= d_{14}\end{aligned}$$

```
z >>= 1
z = 0
```

2.4 Sample Solution

The seven digit constraints (sorted from the most to least significant digit) are:

$$d_1 - 6 = d_{14} \tag{1}$$
$$d_2 - 2 = d_{13} \tag{2}$$
$$d_7 - 1 = d_{12} \tag{3}$$
$$d_8 + 6 = d_{11} \tag{4}$$
$$d_9 + 8 = d_{10} \tag{5}$$
$$d_5 + 1 = d_6 \tag{6}$$
$$d_3 + 7 = d_4 \tag{7}$$

Since model numbers are in base-10 and any valid number does not contain any zero digits, all digits must also satisfy $1 \leq d \leq 9$. Recall that the problem asks for the largest and smallest valid model numbers. This can be done by going through the constraint list and choosing the largest/smallest value for the left-hand (most significant) digit in the given constraint equation, which also fully determines the value of the right-hand digit.

2.4.1 Largest Valid Model Number

Choose the largest possible value for the left-hand digit.

1. $d_1 - 6 = d_{14} \longrightarrow d_1 = 9, d_{14} = 3$
2. $d_2 - 2 = d_{13} \longrightarrow d_2 = 9, d_{13} = 7$
3. $d_7 - 1 = d_{12} \longrightarrow d_7 = 9, d_{12} = 8$
4. $d_8 + 6 = d_{11} \longrightarrow d_8 = 3, d_{11} = 9$
5. $d_9 + 8 = d_{10} \longrightarrow d_9 = 1, d_{10} = 9$
6. $d_5 + 1 = d_6 \longrightarrow d_5 = 8, d_6 = 9$
7. $d_3 + 7 = d_4 \longrightarrow d_3 = 2, d_4 = 9$

$$\boxed{99298993199873}$$

2.4.2 Smallest Valid Model Number

Choose the smallest possible value for the left-hand digit.

1. $d_1 - 6 = d_{14} \longrightarrow d_1 = 7, d_{14} = 1$
2. $d_2 - 2 = d_{13} \longrightarrow d_2 = 3, d_{13} = 1$
3. $d_7 - 1 = d_{12} \longrightarrow d_7 = 2, d_{12} = 1$
4. $d_8 + 6 = d_{11} \longrightarrow d_8 = 1, d_{11} = 7$
5. $d_9 + 8 = d_{10} \longrightarrow d_9 = 1, d_{10} = 9$
6. $d_5 + 1 = d_6 \longrightarrow d_5 = 1, d_6 = 2$
7. $d_3 + 7 = d_4 \longrightarrow d_3 = 1, d_4 = 8$

$$\boxed{73181221197111}$$