Advent of Code 2021, day 24

Examine the inputs

```
In[•]:= input = "inp w
mul x 0
add x z
mod x 26
div z 1
add x 10
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 0
mul y x
add z y
inp w
mul x 0
add x z
mod x 26
div z 1
add x 12
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 6
mul y x
add z y
inp w
mul x 0
add x z
```

```
mod x 26
div z 1
add x 13
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 4
mul y x
add z y
inp w
mul x 0
add x z
mod x 26
div z 1
add x 13
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 2
mul y x
add z y
inp w
mul x 0
add x z
mod x 26
div z 1
add x 14
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 9
mul y x
add z y
inp w
mul x 0
add x z
mod x 26
div z 26
add x -2
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 1
mul y x
add z y
inp w
mul x 0
add x z
mod x 26
div z 1
add x 11
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 10
mul y x
add z y
inp w
mul x 0
add x z
mod x 26
div z 26
add x -15
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 6
mul y x
add z y
inp w
mul x 0
add x z
mod x 26
div z 26
add x -10
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 4
mul y x
add z y
inp w
mul x 0
add x z
mod x 26
div z 1
add x 10
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 6
mul y x
add z y
```

```
inp w
mul x 0
add x z
mod x 26
div z 26
add x -10
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 3
mul y x
add z y
inp w
mul x 0
add x z
mod x 26
div z 26
add x -4
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 9
mul y x
add z y
inp w
mul x 0
add x z
mod x 26
div z 26
add x -1
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 15
mul y x
add z y
inp w
mul x 0
add x z
mod x 26
div z 26
add x -1
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 5
mul y x
add z y";
```

```
In[•]:= parseInt[s_String] :=
         ToExpression[s, StandardForm, Hold] /. {Hold[i_Integer] :→ i, _ → s}
```

```
In[•]:= parsed = (parseInt /@ StringSplit[#, " "]) & /@ StringSplit[input, "\n"];
```

```
In[•]:= runs = Split[parsed, #2 ≠ {"inp", "w"} &];
```

```
In[•]:= runs⟦1 ;; 2⟧
Out[•]= {{{inp, w}, {mul, x, 0}, {add, x, z}, {mod, x, 26}, {div, z, 1}, {add, x, 10},
           {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, 0}, {mul, y, x}, {add, z, y}},
         {{inp, w}, {mul, x, 0}, {add, x, z}, {mod, x, 26}, {div, z, 1}, {add, x, 12},
           {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, 6}, {mul, y, x}, {add, z, y}}}
```

The only places where the instructions differ:

```
In[•]:= Flatten@Position[Equal @@@ Transpose[runs], False]
Out[•]= {5, 6, 16}
```

```
In[•]:= reduced = MapIndexed[{#1⟦5, 3⟧, #1⟦6, 3⟧, #1⟦16, 3⟧, #2⟦1⟧} &, runs]

Out[•]= {{1, 10, 0, 1}, {1, 12, 6, 2}, {1, 13, 4, 3}, {1, 13, 2, 4}, {1, 14, 9, 5},
         {26, -2, 1, 6}, {1, 11, 10, 7}, {26, -15, 6, 8}, {26, -10, 4, 9}, {1, 10, 6, 10},
         {26, -10, 3, 11}, {26, -4, 9, 12}, {26, -1, 15, 13}, {26, -1, 5, 14}}
```

Simplify by hand what happens in that case:

```
In[•]:= fCompressed[a_, b_, c_, state_, zIn_] :=
         With[{x = If[Mod[zIn, 26] + b ≠ w[state], 1, 0]},
          Quotient[zIn, a] (25 x + 1) + (w[state] + c) x
         ]
```

(optional) Verify that the simplified version is the same

```
In[•]:= moreParsed = parsed /. {"add" → Plus, "div" → Quotient,
            "mod" → Mod, "eql" → (Boole[#1 == #2] &), "mul" → Times};

In[•]:= evaluateLonghand[instructions_] :=
         Last[Fold[Function[{state, instruction}, Switch[instruction,
                {"inp", "w"},
                {First[state] + 1, Append[Last[state], "w" → w[First@state]]},
                _, {state⟦1⟧, Append[state⟦2⟧, instruction⟦2⟧ → instruction⟦1⟧[
                    state⟦2⟧[instruction⟦2⟧], If[StringQ[Last@instruction],
                     state⟦2⟧[instruction⟦3⟧], Last@instruction]]]}
              ]], {1, <|"x" → 0, "y" → 0, "z" → 0|>}, instructions]]["z"] //
          FullSimplify[#, Flatten[w[#] ∈ Integers && 1 ≤ w[#] ≤ 9 & /@ Range[13]]] &
```

They do agree, at least for the first seven rounds:

```
In[•]:= evaluateLonghand[moreParsed⟦1 ;; 7 Length@First@runs⟧] ==
          Fold[fCompressed[Sequence @@ #2, #1] &, 0, reduced⟦1 ;; 7⟧] //
         FullSimplify[#, Flatten[w[#] ∈ Integers && 1 ≤ w[#] ≤ 9 & /@ Range[13]]] &

Out[•]= True
```

Solve

I’ve used a bunch of user-defined symbols like if instead of If, to demonstrate that I’m not using any of the Awesome Power of Mathematica. I’m specifically only calling out to Mathematica when there are genuinely numeric quantities to compute with. Notice how I’m not even using Mathematica booleans!

```
In[•]:= f[a_, b_, c_, state_, zIn_] :=
         With[{x = if[not[equal[plus[mod[zIn, 26], b], w[state]]], 1, 0]},
          plus[times[quotient[zIn, a] , plus[times[25, x], 1]],
           times[plus[w[state], c], x]]
         ]

In[•]:= quotient[plus[k_, w[_]], m_] /;
          equal[quotient[plus[k, 1], m], quotient[plus[k, 9], m]] === true :=
 quotient[plus[k, 1], m]
quotient[plus[a_, times[k_?NumericQ , b_]], n_?NumericQ] /;
  n > 1 && IntegerQ@Log[n, k] := plus[quotient[a, n], times[n^(Log[n,k]-1), b]]
quotient[plus[a_, times[b_, k_?NumericQ ]], n_?NumericQ] /;
  n > 1 && IntegerQ@Log[n, k] := plus[quotient[a, n], times[n^(Log[n,k]-1), b]]
quotient[plus[times[b_, k_?NumericQ ], a_], n_?NumericQ] /;
  n > 1 && IntegerQ@Log[n, k] := plus[quotient[a, n], times[n^(Log[n,k]-1), b]]
quotient[plus[times[k_?NumericQ, b_ ], a_], n_?NumericQ] /;
  n > 1 && IntegerQ@Log[n, k] := plus[quotient[a, n], times[n^(Log[n,k]-1), b]]
quotient[a_, 1] := a
quotient[quotient[a_, b_], c_] := quotient[a, times[b, c]]
quotient[plus[w[_], b_], c_] /; less[plus[9, b], c] === true := 0

mod[plus[times[a_ , b_], c_], a_] := c
mod[plus[times[a_ , b_], c_], b_] := c
mod[a_?NumericQ, b_?NumericQ] := Mod[a, b]
mod[w[n_], k_] /; less[9, k] === true := w[n]

positive[x_] := less[0, x]
positive[mod[a_, n_]] /; positive[a] === true := true

nonnegative[w[_]] := true
nonnegative[mod[a_, n_]] /; nonnegative[a] === true := true
nonnegative[if[cond_, trueCase_, falseCase_]] /;
  and[nonnegative[trueCase], nonnegative[falseCase]] === true := true
nonnegative[times[a_,  b_]] /;
  and[nonnegative[a], nonnegative[b]] === true := true
nonnegative[plus[a_, b_]] /; and[nonnegative[a], nonnegative[b]] === true := true
nonnegative[quotient[a_, b_]] /; and[nonnegative[a], less[0, b]] === true := true
nonnegative[k_?NumericQ] := If[NonNegative[k], true, false]

equal[times[a_?(nonnegative[#] === true &) , b_?(nonnegative[#] === true &)], 0] :=
 or[equal[a, 0], equal[b, 0]]
equal[plus[a_?(nonnegative[#] === true &),  b_?(nonnegative[#] === true &)], 0] :=
 and[equal[a, 0], equal[b, 0]]
equal[quotient[a_, n_], 0] := less[a, n]
equal[if[cond_, n_?(positive[#] === true &), 0], 0] := not[cond]
equal[if[cond_, 0, n_?(positive[#] === true &)], 0] := cond
equal[w[_], plus[n_, _?(nonnegative[#] === true &)]] /;
  less[9, n] === true := false
equal[w[_], 0] := false
equal[w[_], k_?NumericQ] /; less[9, k] === true := false
equal[x_?(Head[#] =!= w &), w[n_]] := equal[w[n], x]
equal[a_?NumericQ, b_?NumericQ] := If[a == b, true, false]
equal[w[_], plus[w[_], n_?NumericQ]] /; less[9, n] === true := false

less[plus[_?nonnegative, times[n_,  _?(nonnegative[#] === true &)]], n_] := false
```

```
less[a_?NumericQ, b_?NumericQ] := If[a < b, true, false]
less[quotient[a_, b_], c_] := less[a, times[b, c]]
less[a_, plus[b_, c_]] /; and[less[a, b], less[0, c]] === true := true
less[plus[a_, b_], c_] /; and[not@less[a, c], nonnegative[b]] === true := false
less[times[x_, c_], c_] /; positive[c] === true := equal[x, 0]
less[w[_], k_?NumericQ] /; less[9, k] === true := true
less[0, w[_]] := true

if[true, a_, _] := a
if[false, _, a_] := a
if[not[cond_], t_, f_] := if[cond, f, t]

not[not[x_]] := x
not[false] = true;
not[true] = false;

and[_, false] := false
and[false, _] := false
and[true, x_] := x
and[x_, true] := x

or[_, true] := true
or[true, _] := true
or[false, x_] := x
or[x_, false] := x

plus[a_?NumericQ, b_?NumericQ] := a + b
times[a_?NumericQ, b_?NumericQ] := a b
plus[plus[a_, b_?NumericQ], c_?NumericQ] := plus[a, b + c]
times[0, _] := 0
times[_, 0] := 0
times[x_, 1] := x
times[1, x_] := x
plus[a_, 0] := a
plus[0, a_] := a

times[x_, if[cond_, 0, t_]] := if[cond, 0, times[x, t]]
times[x_, if[cond_, t_, 0]] := if[cond, times[x, t], 0]
```

Let’s go!

```
In[◦]:= condition = equal[Fold[f[Sequence @@ #2, #1] &, 0, reduced], 0]
```

```
Out[◦]=
and[less[
   plus[times[quotient[plus[times[quotient[plus[times[quotient[plus[times[
              plus[times[quotient[plus[times[quotient[plus[times[plus[
                        times[plus[times[plus[times[plus[times[w[1], 26],
                              plus[w[2], 6]], 26], plus[w[3], 4]], 26], plus[
                           w[4], 2]], plus[if[equal[w[6], plus[w[5], 7]],
                           0, 25], 1]], if[equal[w[6], plus[w[5], 7]], 0,
                         plus[w[6], 1]]], plus[if[equal[w[7], plus[
                           mod[plus[times[plus[times[… 1 …], … 1 …],
                               … 1 …], … 1 …], 26], 11]], 0, 25], 1]],
                      if[equal[w[7], … 1 …], 0, … 1 …]], 26],
                    plus[… 1 …, 1]], if[… 1 …]], 26], plus[
                  … 1 …]], if[… 1 …]], … 1 …], … 1 …], 26],
            plus[… 1 …]], … 1 …], 26], plus[… 1 …]],
        … 1 …], 26], plus[… 1 …]],
      … 1 …], 26],
   equal[
    … 1 …]]

large output    show less    show more    show all    set size limit...
```

```
In[◦]:= LeafCount[condition]
Out[◦]= 279 080
```

Well, there's a bunch of cases left in here - some `if` clauses we can't refine because we can't prove what their inputs are. Extract them all, and condition on them.

```
In[◦]:= caseBash[exprs_List] := Flatten[Function[{exprAndRules},
             With[{expr = exprAndRules⟦2⟧, rules = exprAndRules⟦1⟧}, With[
               {cases = MinimalBy[Cases[expr, if[cond_, _, _] ⧴ cond, All], LeafCount]},
               If[cases === {}, {exprAndRules}, Function[{case},
                   {{Append[rules, case], expr /. case → true}, {Append[rules, not@case],
                     expr /. case → false}}]@First@cases]]]] /@ exprs, 1]
        caseBash[expr_] := caseBash[{{{}, expr}}]
```

```
In[◦]:= done = Select[FixedPoint[caseBash, condition], #⟦2⟧ =!= false &] // AbsoluteTiming
Out[◦]= {0.270172, {{{equal[w[6], plus[w[5], 7]], equal[w[8], plus[w[7], -5]],
             equal[w[9], plus[w[4], -8]], equal[w[11], plus[w[10], -4]],
             equal[w[12], w[3]], equal[w[13], plus[w[2], 5]]},
           equal[w[14], plus[w[1], -1]]}}}
```

Let's make it look a bit nicer:

```
In[◦]:= translate = {equal → Equal, plus → Plus};
```

```
In[•]:= translated = Flatten[done⟦2, 1⟧] /. translate

Out[•]= {w[6] == 7 + w[5], w[8] == -5 + w[7], w[9] == -8 + w[4],
         w[11] == -4 + w[10], w[12] == w[3], w[13] == 5 + w[2], w[14] == -1 + w[1]}
```

The final answers

Each of the fourteen variables appears exactly once in the list of constraints:

```
In[•]:= Sort[Sequence @@@ Flatten[Cases[#, _w, All] & /@ translated]] == Range[1, 14]

Out[•]= True
```

Each equation has exactly two variables:

```
In[•]:= AllTrue[translated, Length@Cases[#, _w, All] == 2 &]

Out[•]= True
```

So to maximise, we just maximise the smaller-index variable in each constraint.

```
In[•]:= max = Function[{constraint},
            With[{vars = SortBy[Cases[constraint, _w, All], Sequence @@ # &, 1]},
             With[{toBeLarger = vars⟦1⟧, toBeSmaller = vars⟦2⟧},
              Thread[{vars⟦1⟧, vars⟦2⟧} → First@MaximalBy[Select[Tuples[Range[1, 9], 2],
                   constraint /. {toBeLarger → #⟦1⟧, toBeSmaller → #⟦2⟧} &], First]]
             ]
            ]] /@ translated // Flatten

Out[•]= {w[5] → 2, w[6] → 9, w[7] → 9, w[8] → 4, w[4] → 9, w[9] → 1, w[10] → 9,
         w[11] → 5, w[3] → 9, w[12] → 9, w[2] → 4, w[13] → 9, w[1] → 9, w[14] → 8}

In[•]:= w /@ Range[14] /. max

Out[•]= {9, 4, 9, 9, 2, 9, 9, 4, 1, 9, 5, 9, 9, 8}
```

And to minimise, we minimise the smaller-index variable.

```
In[•]:= min = Function[{constraint},
            With[{vars = SortBy[Cases[constraint, _w, All], Sequence @@ # &, 1]},
             With[{toBeLarger = vars⟦1⟧, toBeSmaller = vars⟦2⟧},
              Thread[{vars⟦1⟧, vars⟦2⟧} → First@MinimalBy[Select[Tuples[Range[1, 9], 2],
                   constraint /. {toBeLarger → #⟦1⟧, toBeSmaller → #⟦2⟧} &], First]]
             ]
            ]] /@ translated // Flatten

Out[•]= {w[5] → 1, w[6] → 8, w[7] → 6, w[8] → 1, w[4] → 9, w[9] → 1, w[10] → 5,
         w[11] → 1, w[3] → 1, w[12] → 1, w[2] → 1, w[13] → 6, w[1] → 2, w[14] → 1}

In[•]:= w /@ Range[14] /. min

Out[•]= {2, 1, 1, 9, 1, 8, 6, 1, 1, 5, 1, 1, 6, 1}
```