Advent of Code 2020

Aaron Kaw

Day 7 - Handy Haversacks

You land at the regional airport in time for your next flight. In fact, it looks like you'll even have time to grab some food: all flights are currently delayed due to **issues in luggage processing**.

Due to recent aviation regulations, many rules (your puzzle input) are being enforced about bags and their contents; bags must be color-coded and must contain specific quantities of other color-coded bags. Apparently, nobody responsible for these regulations considered how long they would take to enforce!

For example, consider the following rules:

```
light red bags contain 1 bright white bag, 2 muted yellow bags.
dark orange bags contain 3 bright white bags, 4 muted yellow bags.
bright white bags contain 1 shiny gold bag.
muted yellow bags contain 2 shiny gold bags, 9 faded blue bags.
shiny gold bags contain 1 dark olive bag, 2 vibrant plum bags.
dark olive bags contain 3 faded blue bags, 4 dotted black bags.
vibrant plum bags contain 5 faded blue bags, 6 dotted black bags.
faded blue bags contain no other bags.
dotted black bags contain no other bags.
```

These rules specify the required contents for 9 bag types. In this example, every `faded blue` bag is empty, every `vibrant plum` bag contains 11 bags (5 `faded blue` and 6 `dotted black`), and so on.

You have a **shiny gold** bag. If you wanted to carry it in at least one other bag, how many different bag colors would be valid for the outermost bag?

In the above rules, the following options would be available to you:

- A `bright white` bag, which can hold your `shiny gold` bag directly.
- A `muted yellow` bag, which can hold your shiny gold bag directly, plus some other bags.
- A `dark orange` bag, which can hold `bright white` and `muted yellow` bags, either of which could then hold your `shiny gold` bag.
- A `light red` bag, which can hold `bright white` and `muted yellow` bags, either of which could then hold your `shiny gold` bag.

So, in this example, the number of bag colors that can eventually contain at least one **shiny gold** bag is **4**.

```
struct Bag
        desc::Symbol
        colour::Symbol
        Bag(desc::Symbol, col::Symbol) = new(desc, col)
end

Bag(desc::AbstractString, col::AbstractString) = Bag(desc |> Symbol, col |> Symbol)

function Bag(bag::AbstractString)
        @assert [letter == ' ' for letter ∈ bag] |> sum == 1
        desc_col = split(bag, ' ')
        @assert desc_col |> length == 2
        Bag(desc_col[1], desc_col[2])
end

mutable struct Rule
        case::Bag
        contain::Vector{Tuple{Integer, Bag}}
end

function Rule(rule::AbstractString)
        rule_split = split(rule, ' ')
        rule_split |> typeof
        @assert rule_split[3] == "bags"
        case = Bag(rule_split[1], rule_split[2])
        rules = rule_split[5:end]
        if rules == ["no", "other", "bags."]
                return Rule(case, [])
        end
        Nrules = length(rules) / 4 |> Integer
        contain = Vector{Tuple{Integer, Bag}}(undef, 0)
        for n ∈ 1:Nrules
                int = tryparse(Int, rules[4n - 3])
                desc = rules[4n - 2]
                col = rules[4n - 1]
                @assert rules[4n][1:3] == "bag"
                push!(
                        contain,
                        (
                                int,
                                Bag(
                                        desc, col
                                )
                        )
                )
        end
        return Rule(case, contain)
end

all_rules = open("puzzle_input.txt") do file
        [Rule(rule) for rule ∈ eachline(file)]
end
```

```
594-element Array{Main.##WeaveSandBox#254.Rule,1}:
 Main.##WeaveSandBox#254.Rule(Main.##WeaveSandBox#254.Bag(:posh, :blue), Tu
ple{Integer,Main.##WeaveSandBox#254.Bag}[(5, Main.##WeaveSandBox#254.Bag(:p
laid, :chartreuse)), (3, Main.##WeaveSandBox#254.Bag(:plaid, :lime))])
```

```
 Main.##WeaveSandBox#254.Rule(Main.##WeaveSandBox#254.Bag(:clear, :teal), T
uple{Integer,Main.##WeaveSandBox#254.Bag}[(2, Main.##WeaveSandBox#254.Bag(:
dotted, :salmon)), (2, Main.##WeaveSandBox#254.Bag(:wavy, :red))])
 Main.##WeaveSandBox#254.Rule(Main.##WeaveSandBox#254.Bag(:faded, :blue), T
uple{Integer,Main.##WeaveSandBox#254.Bag}[(1, Main.##WeaveSandBox#254.Bag(:
dotted, :chartreuse)), (3, Main.##WeaveSandBox#254.Bag(:dim, :bronze))])
 Main.##WeaveSandBox#254.Rule(Main.##WeaveSandBox#254.Bag(:plaid, :black),
Tuple{Integer,Main.##WeaveSandBox#254.Bag}[(5, Main.##WeaveSandBox#254.Bag(
:muted, :beige)), (2, Main.##WeaveSandBox#254.Bag(:pale, :gold)), (3, Main.
##WeaveSandBox#254.Bag(:wavy, :lavender)), (5, Main.##WeaveSandBox#254.Bag(
:dull, :yellow))])
 Main.##WeaveSandBox#254.Rule(Main.##WeaveSandBox#254.Bag(:bright, :cyan),
Tuple{Integer,Main.##WeaveSandBox#254.Bag}[(2, Main.##WeaveSandBox#254.Bag(
:vibrant, :teal))])
 Main.##WeaveSandBox#254.Rule(Main.##WeaveSandBox#254.Bag(:clear, :magenta)
, Tuple{Integer,Main.##WeaveSandBox#254.Bag}[(2, Main.##WeaveSandBox#254.Ba
g(:dim, :chartreuse))])
 Main.##WeaveSandBox#254.Rule(Main.##WeaveSandBox#254.Bag(:muted, :crimson)
, Tuple{Integer,Main.##WeaveSandBox#254.Bag}[(1, Main.##WeaveSandBox#254.Ba
g(:clear, :violet)), (5, Main.##WeaveSandBox#254.Bag(:dark, :coral)), (1, M
ain.##WeaveSandBox#254.Bag(:pale, :salmon)), (3, Main.##WeaveSandBox#254.Ba
g(:light, :red))])
 Main.##WeaveSandBox#254.Rule(Main.##WeaveSandBox#254.Bag(:dotted, :green),
 Tuple{Integer,Main.##WeaveSandBox#254.Bag}[(3, Main.##WeaveSandBox#254.Bag
(:muted, :plum))])
 Main.##WeaveSandBox#254.Rule(Main.##WeaveSandBox#254.Bag(:pale, :crimson),
 Tuple{Integer,Main.##WeaveSandBox#254.Bag}[(3, Main.##WeaveSandBox#254.Bag
(:pale, :maroon)), (2, Main.##WeaveSandBox#254.Bag(:mirrored, :tan))])
 Main.##WeaveSandBox#254.Rule(Main.##WeaveSandBox#254.Bag(:shiny, :black),
Tuple{Integer,Main.##WeaveSandBox#254.Bag}[(1, Main.##WeaveSandBox#254.Bag(
:wavy, :tomato))])
 ⋮@*(Main.(*@##WeaveSandBox#254.Rule(Main.##WeaveSandBox#254.Bag(:dark, :lime), Tu
ple{Integer,Main.##WeaveSandBox#254.Bag}[(1, Main.##WeaveSandBox#254.Bag(:d
ark, :salmon)), (5, Main.##WeaveSandBox#254.Bag(:pale, :green)), (2, Main.#
#WeaveSandBox#254.Bag(:striped, :green)), (5, Main.##WeaveSandBox#254.Bag(:
mirrored, :blue))])
 Main.##WeaveSandBox#254.Rule(Main.##WeaveSandBox#254.Bag(:vibrant, :black)
, Tuple{Integer,Main.##WeaveSandBox#254.Bag}[(3, Main.##WeaveSandBox#254.Ba
g(:light, :red)), (4, Main.##WeaveSandBox#254.Bag(:plaid, :lime)), (3, Main
.##WeaveSandBox#254.Bag(:posh, :lime)), (2, Main.##WeaveSandBox#254.Bag(:do
tted, :lavender))])
 Main.##WeaveSandBox#254.Rule(Main.##WeaveSandBox#254.Bag(:light, :beige),
Tuple{Integer,Main.##WeaveSandBox#254.Bag}[(5, Main.##WeaveSandBox#254.Bag(
:muted, :blue)), (2, Main.##WeaveSandBox#254.Bag(:faded, :red)), (1, Main.#
#WeaveSandBox#254.Bag(:muted, :turquoise))])
 Main.##WeaveSandBox#254.Rule(Main.##WeaveSandBox#254.Bag(:dark, :red), Tup
le{Integer,Main.##WeaveSandBox#254.Bag}[(2, Main.##WeaveSandBox#254.Bag(:fa
ded, :crimson)), (1, Main.##WeaveSandBox#254.Bag(:wavy, :maroon)), (2, Main
.##WeaveSandBox#254.Bag(:clear, :indigo))])
 Main.##WeaveSandBox#254.Rule(Main.##WeaveSandBox#254.Bag(:shiny, :crimson)
, Tuple{Integer,Main.##WeaveSandBox#254.Bag}[(1, Main.##WeaveSandBox#254.Ba
g(:muted, :red)), (5, Main.##WeaveSandBox#254.Bag(:shiny, :bronze)), (1, Ma
in.##WeaveSandBox#254.Bag(:plaid, :green)), (5, Main.##WeaveSandBox#254.Bag
(:mirrored, :orange))])
 Main.##WeaveSandBox#254.Rule(Main.##WeaveSandBox#254.Bag(:dim, :purple), T
uple{Integer,Main.##WeaveSandBox#254.Bag}[(5, Main.##WeaveSandBox#254.Bag(:
striped, :purple)), (2, Main.##WeaveSandBox#254.Bag(:vibrant, :bronze)), (2
, Main.##WeaveSandBox#254.Bag(:striped, :beige))])
 Main.##WeaveSandBox#254.Rule(Main.##WeaveSandBox#254.Bag(:wavy, :maroon),
```

```
Tuple{Integer,Main.##WeaveSandBox#254.Bag}[(1, Main.##WeaveSandBox#254.Bag(
:faded, :silver)), (4, Main.##WeaveSandBox#254.Bag(:muted, :black))])
 Main.##WeaveSandBox#254.Rule(Main.##WeaveSandBox#254.Bag(:dull, :blue), Tu
ple{Integer,Main.##WeaveSandBox#254.Bag}[(2, Main.##WeaveSandBox#254.Bag(:b
right, :gold)), (3, Main.##WeaveSandBox#254.Bag(:pale, :salmon))])
 Main.##WeaveSandBox#254.Rule(Main.##WeaveSandBox#254.Bag(:plaid, :green),
Tuple{Integer,Main.##WeaveSandBox#254.Bag}[(4, Main.##WeaveSandBox#254.Bag(
:dark, :brown)), (1, Main.##WeaveSandBox#254.Bag(:shiny, :orange))])
```

1 Part One

How many bag colors can eventually contain at least one `shiny gold` bag? (The list of rules is quite long; make sure you get all of it.)

1.1 Solution

```
import Base.==
function ==(one::Bag, two::Bag)
        if one.desc ≠ two.desc
                return false
        elseif one.colour ≠ two.colour
                return false
        else
                return true
        end
end

all_bags = Vector{Bag}(undef, 0)
for rule ∈ all_rules
        if !any([rule.case == bag for bag ∈ all_bags])
                push!(all_bags, rule.case)
        end
        for rule_bag ∈ [contains[2] for contains ∈ rule.contain]
                if !any([rule_bag == bag for bag ∈ all_bags])
                        push!(all_bags, rule_bag)
                end
        end
end

all_bag_contains = Vector{Tuple{Bag, Vector{Bag}}}(undef, 0)
for bag_children ∈ all_bags
        bag_parents = Vector{Bag}(undef, 0)
        for rule ∈ all_rules
                if [
                        rule_cont_bag == bag_children for rule_cont_bag ∈ [
                                quant[2] for quant ∈ rule.contain
                        ]
                ] |> any
                        push!(bag_parents, rule.case)
                end
        end
        push!(all_bag_contains, (bag_children, bag_parents))
end

contains_shiny_gold_bag = [Bag("shiny", "gold")]
```

```
Nbagscontain = 0
while Nbagscontain ≠ length(contains_shiny_gold_bag)
        global Nbagscontain = length(contains_shiny_gold_bag)
        for bag_contains ∈ all_bag_contains
                if bag_contains[1] ∈ contains_shiny_gold_bag
                        for bag_container ∈ bag_contains[2]
                                if bag_container ∉ contains_shiny_gold_bag
                                        push!(contains_shiny_gold_bag, bag_container)
                                end
                        end
                end
        end
end
(contains_shiny_gold_bag |> length) - 1 |> println
```

```
142
```

2 Part Two

It's getting pretty expensive to fly these days - not because of ticket prices, but because of the ridiculous number of bags you need to buy!

Consider again your shiny gold bag and the rules from the above example:

- `faded blue` bags contain `0` other bags.
- `dotted black` bags contain `0` other bags.
- `vibrant plum` bags contain `11` other bags: `5` `faded blue` bags and `6` `dotted black` bags.
- `dark olive` bags contain `7` other bags: `3` `faded blue` bags and `4` `dotted black` bags.

So, a single `shiny gold` bag must contain `1` `dark olive` bag (and the `7` bags within it) plus `2` `vibrant plum` bags (and the `11` bags within **each** of those): `1 + 1*7 + 2 + 2*11` = **`32`** bags!

Of course, the actual rules have a small chance of going several levels deeper than this example; be sure to count all of the bags, even if the nesting becomes topologically impractical!

Here's another example:

```
shiny gold bags contain 2 dark red bags.
dark red bags contain 2 dark orange bags.
dark orange bags contain 2 dark yellow bags.
dark yellow bags contain 2 dark green bags.
dark green bags contain 2 dark blue bags.
dark blue bags contain 2 dark violet bags.
dark violet bags contain no other bags.
```

In this example, a single `shiny gold` bag must contain `126` other bags.

How many individual bags are required inside your single `shiny gold` bag?