Advent of Code 2024

Rens Oliemans

December 1, 2024

Contents

Finding the Chief Historian! This program contains my Advent of Code solutions for 2024, which you can find on my sourcehut and GitHub. I believe GitHub doesn't show the results of code blocks, which means that viewing it there might leave you a bit confused.

1 Day 1

1.1 Part 1

We need to reconcile two lists. We get them in the following form:

```
3   4
4   3
2   5
1   3
3   9
3   3
```

And our goal is to find the "distance" between the two lists.

> To find the total distance between the left list and the right list, add up the distances between all of the [sorted] pairs you found.

For the example above, the correct answer is **11**.

My strategy is: convert the input to pairs of numbers, tranpose them (so we have two lists), sort them, tranpose them again (pairs), and take the difference and sum it. Since we might require the input as lists of numbers later separately, we can create a function that parses the input and returns pairs of numbers:

```
(ns aoc.1
  (:require [clojure.string :as str]))

(defn numbers "Converts the puzzle input into pairs of numbers" [input]
  (let [lines (str/split input #"\n")]
    (->> lines
         (map #(str/split % #" +"))
         (map #(map read-string %)))))
```

Verify that it works:

```
(assert (= '((3 4) (4 3) (2 5) (1 3) (3 9) (3 3))
           (numbers testinput)))
```

Now, I'm going to tranpose these lists, sort them, tranpose them again, take the difference, and sum it. Makes sense? We need the two tiny helper functions `sum` and `tranpose`:

```
(defn- sum "Finds the sum of a vector of numbers" [vec]
  (reduce + vec))

(defn- transpose "Tranposes a matrix" [m]
  (apply mapv vector m))
```

With the final function being now quite easy to follow if you keep my strategy above in mind. Recall that the correct answer for the testinput was 11.

```
(defn p1 [input]
  (let [input (numbers input)]
    (->> input
         (transpose)
         (map sort)
         (transpose)
         (map #(abs (- (first %) (second %))))
         (sum))))

(p1 testinput)
```

```
11
```

It works for the testinput, fantastic. Now let's open the file and run it on the input. The input file for day 1 can be found in the file `inputs/1`.

```
(def input (slurp "inputs/1"))
(p1 input)
```

```
2057374
```

Hurrah! We get a **Gold Star**! See Part 2

1.2 Part 2

Now, we need to find a "similarity score" for the two lists:

> Calculate a total similarity score by adding up each number in the left list after multiplying it by the number of times that number appears in the right list.

A naïve way to do this would be to iterate over the first list, where, for each element, we count how many items in the second list are equal to that element, and multiply the element with the count. However, you'd be doing a lot of duplicate counting. A faster way to do it is to convert the second (it doesn't really matter which one you pick) list to a map once, with `{element frequency}`. Let's use the function `frequencies`!

```
(frequencies (last (transpose (numbers testinput))))
```

```
{4 1, 3 3, 5 1, 9 1}
```

Now, we can iterate over the first list (which we get by `(tranpose (numbers input))`), multiply the element itself by the count in `frequencies`, and sum the result.

```
(defn p2 [input]
  (let [input (transpose (numbers input))
        one (first input)
        freqs (frequencies (second input))]
    (->> one
         (map #(* % (freqs % 0)))
         (sum))))

(assert (= 31 (p2 testinput)))
(p2 input)
```

```
23177084
```