Ranges of seeds

Johan Montelius

Spring Term 2023

Introduction

This assignment is based on the Advent of Code 2023 day 5 problem. The first task is easier and the second requires some more coding. For the first task you can choose you own method but for the second task you you should solve the problem using an implementation of ranges.

The first task

The first part of the puzzle is quite simple. As input you're given four seeds and a seven transformation maps. Each map tells you how seed numbers are transformed into soil number, soil numbers to fertilizer number etc. A transformation is describes as: *destination*, *source* and *range*. A sequence: `50 98 2` means that number 98 and 99 should be transformed to 50 and 51. You task is to find the lowest *location number* of the seeds.

Given below is a sample problem that you can work with before trying the true task.

```
seeds: 79 14 55 13

seed-to-soil map:
50 98 2
52 50 48

soil-to-fertilizer map:
0 15 37
37 52 2
39 0 15

fertilizer-to-water map:
49 53 8
0 11 42
42 0 7
```

```
57 7 4

water-to-light map:
88 18 7
18 25 70

light-to-temperature map:
45 77 23
81 45 19
68 64 13

temperature-to-humidity map:
0 69 1
1 0 69

humidity-to-location map:
60 56 37
56 93 4
```

If we take seed nr 79 as an example it will require *soil* nr 81 (since all nr 50..98 are mapped to 52..100 i.e. two numbers more. Soil nr 81 is then mapped to fertilizer 81 (since there is no special rule). We then end up with water nr 81 but light nr 74. The temperature change to nr 78. This is also the humidity but the location is 82.

You first job is to parse the input string and turn it in to something more convenient to work with. To do this you could work with the following functions:

- `File.read!("sample.txt")`: assuming the sample is stored in the file `sample.txt` the function returns a string with the text.
- `String.split(text, divider)` : if the text is our original string we could first split it using a divider `"\n\n"` i.e double new-line. The function will return a list of the different segments: the seeds and the different maps. The string that represents the seeds can then be further divided using `" "` as the divider etc.
- `Integer.parse(string)` : this function will parse a string and return, if successful, a tuple `{nr, rest}` where `nr` is a number and `rest` is what ever came after the number. The string has to start with a digit so the string `" 20"` will not work.
- `String.trim(string)` : this function will return a string where all space, newline etc has been remove from the beginning and end of the string. This could come in handy before trying to parse an integer. It

could also be a way of removing a trailing newline on the full sample text (or you might end up with an empty row).

The file "sample.txt" of course needs to be in the same directory where you are when you execute the program. If you run your Elixir shell from command line there is little discussion of where this is but if you're using an IDE you have to think a bit.

When you parse the sample string you will of course return a data structure that represents the problem. To make things easier you can assume that the maps are always written in the same order.

Once you have the initial seed numbers and maps you should implement a function that applies the transformations. If you get the transformations right the location numbers for each of the seeds will be: 82, 43, 86 and 35 so the lowest one is 35. If you pass this test take the real task and give it a try; what is the lowest location number?

Describe your implementation and justify your choice of data structures.

The second task

The input data is the same as in the first task but the interpretation has changed. The first row in the specification does not specify four seeds but **two ranges of seeds**, 79..92 and 55..67). The seeds are specified as the first seed number and how many consecutive seed numbers we have (79 14 means 79,80,81...92).

You can use your original program and instead of exploring the resulting location of four seeds you explore 14 + 13 = 27 seeds. This works fine, give it a try.

You think that everything works fine and then you try not the sample but the real problem..... ops. Your program will either crash or run for ever, the number of seeds to try are millions and millions, something has to be done.

Ranges

In Elixir you have ranges of integers built in. The range of all integers from 12 to 18 is written `12..18` and you can then use this data structure as input to all functions in the `Enum` module. This is great but the range data structure lacks some functionality. The ranges must be consecutive so you can not talk about the range `1..5 - 8..12` etc.

You task is to implement a new data structure that represent a sequence of ranges. We could for example have the sequence: 1..5, 11..13 and 18..24. This sequence then includes for example 2, 4 and 19 but not 6,8 nor 15 or 27. It is up to you how this sequence should be represented but you should

also implement some functions over sequences so you might take a look at them before you decide.

The functions that you should implement will be very useful to when we solve the puzzle. The functions are:

- `empty()` : return an empty sequence.
- `range(from,to)` : return a sequence with one range including all number from *from* to *to*.
- `union(a, b)` : given two sequences, return the *union* i.e. a number is part of the union of a and b if, and only if, the number is part of a or b.
- `intersection(a, b)` : given two sequences return the *intersection* i.e. a number is part of the *intersection* if, and only if, it is part of a and of b.
- `difference(a, b)` : given two sequences return the *difference* i.e. a number is part of the *difference* if, and only if, it is part of a but not of b.
- `add(a, n)` : given a sequence, a, and a number n, return a sequence where each element in the original sequence has been incremented by n.

Depending on how you choose to represent a range, it becomes more or less easy to implement these functions. We could of course choose to represent a range `3..5 - 8..10` as list of integers,

$$3, 4, 5, 8, 9, 10$$

, but that might be an impractical form if we are dealing with ranges such as `1..12345678`.

Note - you don't have to, nor might it be very efficient, to use the Elixir representation of ranges for the segments.

Choosing the right representation should be done with the operations in mind; some things might be easier to do depending on the representation.... and, have in mind that the sequences can span over very large sets. Describe your implementation and justify the representation that you have chosen.

The second task revisited

What will happen if you rewrite your program to use sequences of ranges instead of unique numbers? Using the sample you would start with the ranges 55..67 and 79..92. These ranges should now be transformed using

the different maps. A map of course consists of a set of ranges and how those ranges should be transformed.

Take the first map as an example. We see that the range 50..98 should be transformed by adding 2. Since both our seed ranges are fully covered by this range we should end end up with the ranges 57..69 and 81..94. The second mapping does not change the ranges but when going from fertilizer to water we end up with the ranges 53..56, 61..69 and 81..94.

What is the general rule of the transformation? Assume that you have a range *source* that should be incremented by n and a sequence *seeds*, then ... yes, what should be done? If you can describe what should be done in terms of: *union*, *intersection*, *difference* and *add* then you have your implementation.

Start by rewriting the parser, or work with the parsed representation, and produce a new representation of the puzzle. The seeds are now a sequence of ranges and each rule is a set of ranges with rules of transformation.

Using the real input for seeds and maps, What is now the minimum location nr that we end up with? Describe your implementation and the results you obtain.