Springs: Part 1

Ayda Ghalkhanbaz

Spring Term 2024

Introduction

In this assignment we are going to find an algorithm to solve the day 12 of the Advent of Code puzzle. To solve this puzzle, we will find an algorithm that counts the number of possible ways of arranging damaged and working springs together so that the number of damaged springs matches the given number pattern.

Method

The program can be divided into 2 main parts. The first part is about reading an input file (consisting of multiple rows where the description and number sequence are separated by a space in each line), splitting the rows so that each row is in a list and furthermore each line is divided into two further lists; description and numbers, and finally parsing these lists at the end. This process is implemented using a `parse/1` function and it returns 2 parsed and splitted lists (description and numbers) as a tuple.

The second part is the main algorithm where we actually find the solution using recursion. The first function in this part is called `count_combos/2` and it handles different cases. The first case is when the first spring in the description is a working spring (represented by atom `:op`), in this case we don't care about that spring in particular and continue until we reach a damaged (`:dam`) or an unknown spring (`:unk`). In case of an unknown spring, we call the `count_combos/2` function recursively and check two alternatives: how many solutions there are if we replace `:unk` with `:op` or `:dam`.

The more interesting part is when the spring is `:dam`. In this case, the `count_combos/2` function calls a helper function `check_next/2` to check whether the number of `:dam` springs in row are the same as the matched number in the pattern.

For a better understanding of the algorithm, the `count_combos/2` and the `check_next/2` functions are demonstrated in the code snippet below.

```elixir
def count_combos([], []) do 1 end
def count_combos([], _)  do 0 end
def count_combos([:dam|_], [])  do 0 end
def count_combos([:dam|rest], [nr|numbers]) do
  case check_next(rest, nr-1) do
    {:ok, rest} -> count_combos(rest, numbers)
    :nil -> 0
  end
end
def count_combos([:op|rest], numbers) do
  count_combos(rest, numbers)
end
def count_combos([:unk | rest], numbers) do
  count_combos([:dam | rest], numbers) +
      count_combos([:op | rest], numbers)
end
def count_combos(_,_) do 0 end

def check_next([], 0) do {:ok, []} end
def check_next([:op|rest], 0) do {:ok, rest} end
def check_next([:dam|rest], nr) do check_next(rest, nr-1) end
def check_next([:unk|rest], 0) do {:ok, [:op|rest]} end
def check_next([:unk | rest], nr) do check_next(rest, nr - 1) end
def check_next(_,_) do :nil end
```

Result

To test the functionality of the program, an input data was given through the assignment that is expected to have 21 possible solutions in total. Below you'll find the input data and the result of the program test.

Example of input data

```
???.### 1,1,3
.??..??...?##. 1,1,3
?#?#?#?#?#?#?#? 1,3,1,6
????.#...#... 4,1,1
????.######..#####. 1,6,5
?###???????? 3,2,1
```

Run Test of the program

```
iex(71)> Springz.test()
21
```
