Train Shunting

Lucas Larsson

Spring Term 2022

Introduction

Not much of an introduction here, the assignment was straight froward, this report will be mostly about the implementation section.

Implementation

List processing

I followed the instructor tip and started with list processing methods, it was pretty easy, most of the functions had corresponding build in method in the `Enum` module. see code below.

```
    def take(list, n) do
        Enum.take(list,n)
    end

    def drop(list, n) do
        Enum.drop(list,n)
    end

    def position([], _) do IO.puts("element does not exist") end
    def position([h|t], y) do
        cond do
        h == y -> 1                     # if it is the first element return 1
        true -> 1 + position(t, y)      # otherwise continue traversing and incriminating til
      end
  end
```

I won't explain the methods as i believe they are clear and fairly readable for my class mates.

Moves

As instructed I created a new module and called it `moves`, in it are the `single/2` methods.

```
def single({:one, n}, {main, one, two}) do
      cond do
          length(main) - n < 0 ->
              {main, one, two}
          n < 0 ->
              {Train.append(main, Train.take(one, n * -1)),
              Train.drop(one, n * -1), two}
          n > 0 ->
              {Train.take(main, length(main)-n),
              Train.append(Train.drop(main, length(main)-n), one), two}
          true -> {main, one, two}
      end
  end
```

The single methods used to apply moves.

```
// taken from the instructions
For example, single({:one,1},{[:a,:b],[],[]}) returns {[:a],[:b],[]}.
```

Depending on if `n` is smaller or grater than 0 a different clause will run, and that is due to the rules, {`:one, 1`} will result in an addition to the track `:one`, while{`:one, -1`}, will result in moving from `:one` to `main`.

This achieved through first selecting the part to be moved with the `take/2` function, and then append it to a list and drop it from the other depending as mentioned before on the direction of the move. Another `single/3` clause is created for the `:two` track.

```
def move([], state) do [state] end

def move([h|t], state) do
[state | move(t, single(h, state))]
end
```

Later on I wrote this `move/2` procedure that follows the instructions:

- Recursively apply each move in turn.
- When no moves are to be applied, our specification says that the initial state is to be returned (base-case).
- For each move, use single.

Reflection

I followed instructions and done the assignment off course, but this is what fits in 2 pages, over all i think it was a fun assignment, a bit mind boggling at the beginning, but fairly easy by just following the instructions. I realize that i haven't done a lot of analysis or in depth explanation of the assignment but i think it is fine here, because we was nothing to bench mark performance wise and no algorithm analysis.