Exercise set 4: Programming in Erlang

Sequential Programming

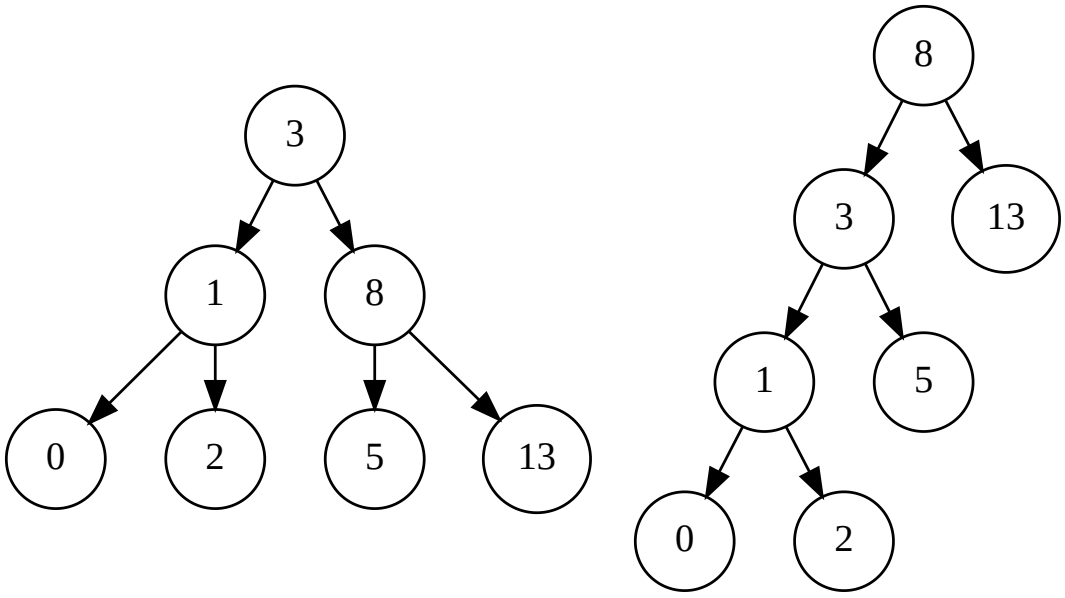
- Finish the following declaration of the function `move` from the Erlang intro slides, so that a position also can be moved to in the directions `south` and `east` (at least).

```
move(north, {X, Y}) -> {X, Y+1};
move(west, {X, Y}) -> {X-1, Y}.
```

- Extend your definition of `move` so that it throws an exception if either the X or Y coordinate becomes negative. (And feel free to improve on the information given with the thrown exception.)
- Make an Erlang module, `bintree`, to work with dictionaries represented as binary search trees. That is, you should provide an API for at least:
  - Create an empty dictionary
  - Insert a key-value pair into a dictionary
  - Find the value associated with a given key
- Make a module, `arith`, to work with abstract syntax trees for arithmetic expressions.

Concurrent Programming

- There can be many different binary trees with the same sequence of values stored in nodes. For example, here are two binary trees storing the sequence 0, 1, 2, 3, 5, 8, 13:



Write a function, `eq_tree`, that determines whether two trees contains the same values when traversing the trees in inorder sequence.

Try to implement `eq_tree` with different strategies, for instance:

- a completely sequential version, where you construct a list for each tree and then compare the two lists;
- a version where you start two processes that each traverse a tree in inorder sequence and send the values to a third process that check whether the two sequences are the same.

What are the pros and cons of your different solutions (memory usage, lines of code, ...). Can you make a version that will stop early? That is, if two trees are *not* equal, then you should only travel as much of the trees that are needed to determine that.

- Make a small string reversal server. That is, a server you can send a string and then it will return the reversed string.
- Change the move server, from the Erlang intro slides, so that it keep track of the current position. (Maybe it's then a position server rather than a move server.)

Asynchronous programming

The Haskell package async (https://hackage.haskell.org/package/async) provide a (monadic) library for running IO operations asynchronously and waiting for their results. Implement an Erlang module, `async`, to provide similar functionality. That is, you should implement the following minimal API, where `Aid` stands for an action ID, which is an opaque datatype (that is, you decide what it should be):

- `new(Fun, Arg)` that starts a concurrent computation that computes `Fun(Arg)`. It returns an action ID.
- `poll(Aid)` that check whether an asynchronous action has completed yet. If it has not completed yet, then the result is `nothing`; otherwise the result is either `{exception, Ex}` if the asynchronous action raised the exception `Ex`, or `{ok, Res}` if it returned the value `Res`.
- `wait(Aid)` that waits for an asynchronous action to complete, and return the value of the computation. If the asynchronous action threw an exception, then the exception is re-thrown by `wait`.

Morse Code

[Rephrased from ruby quiz #121 (http://www.rubyquiz.com/quiz121.html)]

Morse code is a way to encode telegraphic messages in a series of long and short sounds or visual signals. During transmission, pauses are used to group letters and words, but in written form the code can be ambiguous.

For example, using the typical dot (`.`) and dash (`-`) for a written representation of the code, the word `...---..-....-` in Morse code could be an encoding of the names Sofia or Eugenia depending on where you break up the letters:

```
...|---|..-.|..|.-    Sofia
.|..-|--.|.|-.|..|.-  Eugenia
```

We will only focus on the alphabet for this quiz to keep things simple. Here are the encodings for each letter:

```
A .-            N -.
B -...          O ---
C -.-.          P .--.
D -..           Q --.-
E .             R .-.
F ..-.          S ...
G --.           T -
H ....          U ..-
I ..            V ...-
J .---          W .--
K -.-           X -..-
L .-..          Y -.--
M --            Z --..
```

- Declare a function, `encode`, that takes a string (of letters) as argument and translate it to morse code (as a string).
- Declare a function, `decode`, that takes a string encoded in morse code as argument and return a list of all possible translations.