Erlang Assignment

TCSS 380 – Spring 2024

Part 1

Description

You have been hired by a cyber security firm as your first job upon graduation. The team you join are Erlang fanatics and only allow team members to program in this wonderous language. Your first responsibility is a difficult task. Your organization has intercepted a transmission from antagonist parties. The transmission contains a list of numbers. It is assumed that each of these numbers indicate when the next cyber-attack will occur! Your job is to write an algorithm that will identify the smallest number, thereby indicating how long until the next attack.

Here's the kicker, each number is in an *unknown radix* (base). To identify the smallest number in the set, you will need to *identify what is the lowest possible radix* for each individual number, then convert it to a common radix.

Assumptions

- For each number:
  - The max radix possible in this set is base 36.
  - the representation of the symbols [0-9] retain the original value as the decimal numbering system. For example, the value for the symbol ‘7’ is decimal integer 7.
  - the representation of the symbols [a-z] are an *extension* to hexadecimal. For example:

symbol	decimal integer
‘a’	10
‘b’	11
…	…
‘f’	15
‘g’	16
…	…
‘z’	35

Given

- The intercepting team stored the transmission in a file. They will provide:
    - the a sample file called **`sample.txt`** containing the numbers provided in the example below. It is known that all of the attacks will occur in the future so there will be **no negative values** in the file. Your implementation may assume all values will be positive in every case.
    - an Erlang module called **`countdown`** containing an exported function **`readTransmission`**:
        - accepts the file name as a string as a parameter.
        - evaluates to a list containing the numbers in the unknown radix.
            - Each number will be represented as a string. *Remember: string is not a data type in Erlang, instead a string “hello” is shorthand for the list [$h, $e, $l, $l, $o], that is, [104, 101, 108, 108, 111].
            - The symbols in the strings will come from the set of symbols [0-9] and [a-z] (lowercase Latin alphabet)
        - Do NOT alter this function in anyway, use it to obtain your starting values.
        - See **`countdownTest.erl`** for an example usage of **`readTransmission`**.

Requirements

Implementation

I will use Unit testing while grading your submission. As such, you must export the following functions from the **`countdown`** module. For full credit, the API described in the next section must be precisely followed. You are free to use as many non-exported helper functions as needed. Your solution is not required to *use* each exported function described in the API, yet you are required to implement functions.

Unit Tests

In a separate module, write enough unit test cases to gain a high level of confidence that your solution is correct. At a minimum, write individual unit test cases to fully test the values and final solution of the input file **`sample.txt`**. Note, this sample file does not fully cover the range of possible edge cases. You may want to create your own input files and test them accordingly.

API

howLongDoWeHave(L)

Compares a list of numbers with unknown-differing radii and returns the smallest numerical value as a binary number represented as a list containing the individual bits in order.

Parameters:
L – a list of numbers with unknown-differing radii. Each number is represented as an Erlang string.

Example:
[“abc”, “12321”, “789”, “1face”, “z1”] % The smallest value is 12321 (Base 4)

Returns:
the smallest numerical value as a binary number represented as a list containing the individual bits in order.

Example:
[1,0,0,1,1,1,0,1,1]

discoverBase(N)

Returns the radix of the number N.

Parameters:
N – a number with an unknown radix represented as an Erlang string.

Example:
“12321”

Returns:
the radix of the number as an Erlang Number.

Example: **4**

generateBinaryConverter(R)

Returns a function that accepts a number in radix **R** represented as an Erlang string and when evaluated returns the number in binary represented as a list containing the individual bits in order.

Parameters:
R – the radix of the numbers to be converted into binary represented as an Erlang Number.

Example:
4

Returns:
A function F as described below:

F(N)

Returns the number **N** in binary represented as a list containing the individual bits in order.

Parameters:
N – a number in the radix of this function represented as an Erlang string.

Example:
“12321”

Returns:
The number in binary represented as a list containing the individual bits in order

Example: **[1,0,0,1,1,1,0,1,1]**

```
4> F = generateBinaryConverter(4).
#Fun<countdown.1.23232332>
5> F(“12321”).
[1,1,0,1,1,1,0,1,1]
```

Part 2

Description

Great, now that the organization has a function that predicts the next cyber-attack, security analysts need to perform operations on the resultant representation of the binary number. You performed well in your first task, so you are assigned to create a new module that exports multiple functions that perform binary operations.

Requirements

Implementation

Create a new module called **`binaryOps`**. I will use Unit testing while grading your submission. As such, you must export the following functions from the **`binaryOps`** module. For full credit, the API described in the next section must be precisely followed.

Notes

- This representation of binary numbers:
  - does not account for negative values. Do not consider a signed bit or other negative representations.
  - uses the empty list [] to represent the value for 0. The reasoning for this is that when leading 0s are trimmed, [] is left for 0.
  - only considers integers.

Unit Tests

In a separate module, write enough unit test cases to gain a high level of confidence that your solution is correct.

API

binNegate(BinL)

Performs and returns logical negation on each bit, forming the ones' complement of the given binary value and returns the result. Bits that are 0 become 1, and those that are 1 become 0.

Parameters:
`BinL` – a binary number represented as a list containing the individual bits in order. All leading (left) 0s are ignored.

Example:
`[1,0,0,1,1,1,0,1,1]`

Returns:
the negation of the argument as a binary number represented as a list containing the individual bits in order. The resulting list must not contain leading (left) 0s.

Example:
`[1,1,0,0,0,1,0,0]`

binAnd(BinL1, BinL2)

Performs and returns the logical AND operation on each pair of the corresponding bits and returns the result. Thus, if both bits in the compared position are 1, the bit in the resulting binary representation is 1 otherwise, the result is 0.

Parameters:
`BinL1` – a binary number represented as a list containing the individual bits in order.

`BinL2` – a binary number represented as a list containing the individual bits in order.

Example:
`[1,0,0,1,1,1,0,0,1] [1,1,0,0]`

Returns:
the logical AND operation on each pair of the corresponding bits between each argument as a list containing the individual bits in order. The resulting list must not contain leading (left) 0s.

Example:
`[1,0,0,0]`

binOr(BinL1, BinL2)

Performs and returns the logical OR operation on each pair of the corresponding bits and returns the result. Thus, if either bits in the compared position are 1, the bit in the resulting binary representation is 1 otherwise, the result is 0.

Parameters:
BinL1 – a binary number represented as a list containing the individual bits in order.

BinL2 – a binary number represented as a list containing the individual bits in order.

Example:
[1,0,0,1,1,1,0,0,1] [1,1,0,0]

Returns:
the logical OR operation on each pair of the corresponding bits between each argument as a list containing the individual bits in order. The resulting list must not contain leading (left) 0s.

Example:
[1,0,0,1,1,1,1,0,1]

binXor(BinL1, BinL2)

Performs the logical XOR operation on each pair of the corresponding bits and returns the result. The result in each position is 1 if only one of the bits is 1, otherwise, the result is 0.

Parameters:
BinL1 – a binary number represented as a list containing the individual bits in order.

BinL2 – a binary number represented as a list containing the individual bits in order.

Example:
[1,0,0,1,1,1,0,0,1] [1,1,0,0]

Returns:
the logical XOR operation on each pair of the corresponding bits between each argument as a list containing the individual bits in order. The resulting list must not contain leading (left) 0s.

Example:
[1,0,0,1,1,0,1,0,1]

binAdd(BinL1, BinL2)

Performs an addition operation (a + b) on the arguments and returns the result.

Parameters:
`BinL1` – a binary number represented as a list containing the individual bits in order.

`BinL2` – a binary number represented as a list containing the individual bits in order.

Example:
`[1,1,0,0,1] [1,1,0,0]`

Returns:
the result of performing an addition (a + b) operation of the arguments. The resulting list must not contain leading (left) 0s.

Example:
`[1,0,0,1,0,1]`

trimLeading(BinL)

Returns the result of removing all leading(left) 0s from the argument.

Parameters:
`BinL` – a binary number represented as a list containing the individual bits in order.

Example:
`[0,0,0,1,1,1,0,0,1]`

Returns:
the result of removing all leading(left) 0s from the argument as a list containing the individual bits in order.

Example:
`[1,1,1,0,0,1]`