1 Task I.1 — Hamming Numbers

1.1 Solutions

For this problem I developed 2 solutions, one using a naive approach checking which numbers are hamming numbers and the other generating the list of hamming numbers based upon previous values in the list.

1.1.1 Naive Approach

The Naive approach is defined in the following way:

```
import Data.Numbers.Primes
```

The type of hamming' is a list elements of the type class Integral. Strictly it is of type Integer but the use of type classes allows for easy readability and expandability. hamming' is an infinite list of Integrals.

hamming' generates an infinite list of all integers from 1 onwards and then filters the list based upon a wether the number is a hamming number. This filter is done by checking each number in the list against a condition which expresses if a number is a hamming number. This condtion is expressed in the lambda function.

The test for hamming numbers exploits the fact that hamming numbers are created by multiplying another hamming number by 2, 3 or 5. This means that any hamming number can be represented as $2^i 3^j 5^k \; where \; x, y, z \in Z$. As 2, 3 and 5 are also prime numbers it means that this representation this is also the prime factorisation of the hamming number. Therefore any hamming number will have only 2, 3 and 5 as prime factors.

To exploit this the condition to filter on finds the prime factors of the number (represented as the list of prime factors) and checks that the only elements in the prime factor list are either 2, 3 or 5. If there is any prime factors that aren't 2, 3 or 5 then the number isn't added to the list and the next value is tested.

The problem with this approach is that it tests all the integers systematically meaning that there is a lot of numbers to consider. primeFactors also takes a long time to run as it has to systematically find the prime factors. This gets computational expensive for large numbers.

1.1.2 Main Approach

The main approach is defined in the following way:

```
hamming :: Integral a => [a]
hamming = 1 : (merge hamming5 $ merge hamming2 hamming3)
              where hamming2 = map (2*) hamming
                    hamming3 = map (3*) hamming
                    hamming5 = map (5*) hamming
```

The type of hamming is the same as hamming' as they will both produce the same infinite list. This being a list of Integrals.

EXPLAIN THIS

1.2 Cyclic Graphs

2 Task I.2 — SpreadSheet Evaluator

3 Task I.3 — Skew Binary Random Access Trees

4 Task I.4 — Interval Arithmetic