THE KEPLER BINARY TREE OF REDUCED FRACTIONS

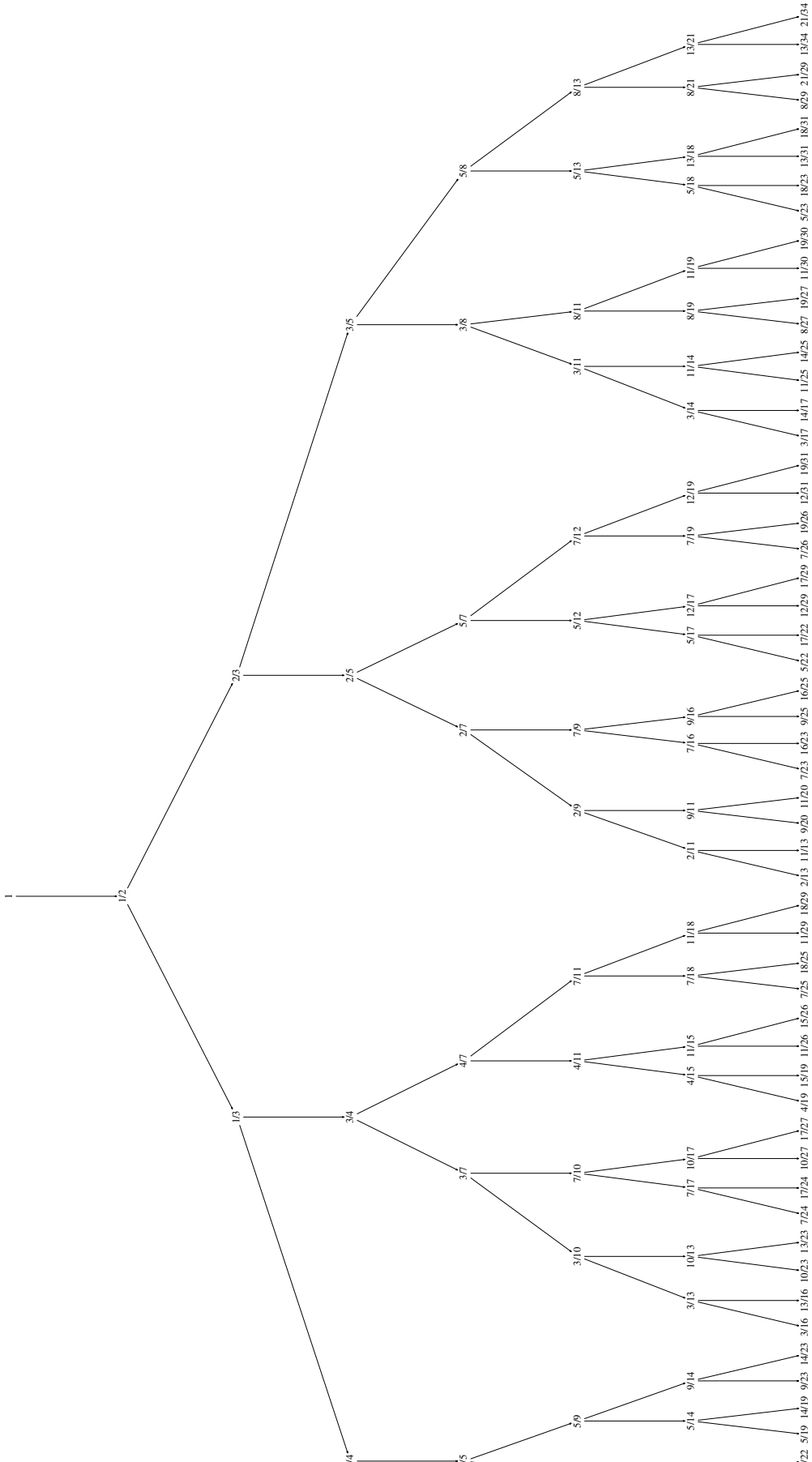
RICHARD J. MATHAR

Abstract. The reduced fractions in the interval $(0, 1)$ appear in Johannes Kepler's 'Harmonices Mundi' of 1619, see sequence A294442 of the Online Encyclopedia of Integer Sequences. This here is some brainstorming on the question how many different numerators appear at increasing levels of this binary tree of fractions.

1. Directed Graph of Fractions

1.1. **Definition.** We define a directed graph with nodes of fractions i/j by placing the fraction 1/1 at the top and generating for each node (except the top node) two children representing the fractions $i/(i+j)$ and $j/(i+j)$ [3][5, Ex. 3.4][4, p. 36][2, Fig. 11][7, A294442]. This graph is displayed on the next page:

Date: December 2, 2017.


1
1/2
1/3
2/3
1/4
3/4
2/5
3/5
1/5
4/5
2/7
3/7
4/7
5/7
1/7
6/7
5/8
3/8
7/8
1/8
2/9
7/9
4/9
5/9
7/10
3/10
7/11
4/11
1/11
6/11
5/11
10/11
9/11
8/11
3/11
11/12
7/12
5/12
1/12
11/13
8/13
5/13
2/13
12/13
9/13
10/13
3/13
13/14
11/14
9/14
5/14
1/14
6/14
8/14
12/14
13/15
11/15
4/15
14/15
1/15
7/15
2/15
8/15
14/16
9/16
7/16
5/16
3/16
15/16
1/16
11/16
13/16
10/16
12/16
6/16
4/16
2/16
15/17
12/17
5/17
13/17
8/17
3/17
16/17
1/17
14/17
11/17
6/17
9/17
4/17
7/17
10/17
16/18
13/18
5/18
17/18
1/18
7/18
11/18
15/18
14/18
12/18
10/18
8/18
6/18
4/18
2/18
17/19
15/19
13/19
11/19
9/19
7/19
5/19
3/19
1/19
18/19
16/19
14/19
12/19
10/19
8/19
6/19
4/19
2/19
19/20
17/20
15/20
13/20
11/20
9/20
7/20
5/20
3/20
1/20
18/20
16/20
14/20
12/20
10/20
8/20
6/20
4/20
2/20
19/21
17/21
15/21
13/21
11/21
9/21
7/21
5/21
3/21
1/21
20/21
18/21
16/21
14/21
12/21
10/21
8/21
6/21
4/21
2/21
19/22
17/22
15/22
13/22
11/22
9/22
7/22
5/22
3/22
1/22
20/22
18/22
16/22
14/22
12/22
10/22
8/22
6/22
4/22
2/22
21/22
19/23
17/23
15/23
13/23
11/23
9/23
7/23
5/23
3/23
1/23
22/23
20/23
18/23
16/23
14/23
12/23
10/23
8/23
6/23
4/23
2/23
21/24
19/24
17/24
15/24
13/24
11/24
9/24
7/24
5/24
3/24
1/24
22/24
20/24
18/24
16/24
14/24
12/24
10/24
8/24
6/24
4/24
2/24
21/25
19/25
17/25
15/25
13/25
11/25
9/25
7/25
5/25
3/25
1/25
24/25
22/25
20/25
18/25
16/25
14/25
12/25
10/25
8/25
6/25
4/25
2/25
23/26
21/26
19/26
17/26
15/26
13/26
11/26
9/26
7/26
5/26
3/26
1/26
24/26
22/26
20/26
18/26
16/26
14/26
12/26
10/26
8/26
6/26
4/26
2/26
23/27
21/27
19/27
17/27
15/27
13/27
11/27
9/27
7/27
5/27
3/27
1/27
26/27
24/27
22/27
20/27
18/27
16/27
14/27
12/27
10/27
8/27
6/27
4/27
2/27
25/28
23/28
21/28
19/28
17/28
15/28
13/28
11/28
9/28
7/28
5/28
3/28
1/28
26/28
24/28
22/28
20/28
18/28
16/28
14/28
12/28
10/28
8/28
6/28
4/28
2/28
25/29
23/29
21/29
19/29
17/29
15/29
13/29
11/29
9/29
7/29
5/29
3/29
1/29
28/29
26/29
24/29
22/29
20/29
18/29
16/29
14/29
12/29
10/29
8/29
6/29
4/29
2/29
27/30
25/30
23/30
21/30
19/30
17/30
15/30
13/30
11/30
9/30
7/30
5/30
3/30
1/30
28/30
26/30
24/30
22/30
20/30
18/30
16/30
14/30
12/30
10/30
8/30
6/30
4/30
2/30
27/31
25/31
23/31
21/31
19/31
17/31
15/31
13/31
11/31
9/31
7/31
5/31
3/31
1/31
30/31
28/31
26/31
24/31
22/31
20/31
18/31
16/31
14/31
12/31
10/31
8/31
6/31
4/31
2/31
29/32
27/32
25/32
23/32
21/32
19/32
17/32
15/32
13/32
11/32
9/32
7/32
5/32
3/32
1/32
30/32
28/32
26/32
24/32
22/32
20/32
18/32
16/32
14/32
12/32
10/32
8/32
6/32
4/32
2/32
29/33
27/33
25/33
23/33
21/33
19/33
17/33
15/33
13/33
11/33
9/33
7/33
5/33
3/33
1/33
32/33
30/33
28/33
26/33
24/33
22/33
20/33
18/33
16/33
14/33
12/33
10/33
8/33
6/33
4/33
2/33
31/34
29/34
27/34
25/34
23/34
21/34
19/34
17/34
15/34
13/34
11/34
9/34
7/34
5/34
3/34
1/34
32/34
30/34
28/34
26/34
24/34
22/34
20/34
18/34
16/34
14/34
12/34
10/34
8/34
6/34
4/34
2/34

Definition 1. *We call the fraction $i/(i+j)$ the 0-child and the fraction $j/(i+j)$ the 1-child of the fraction i/j.*

Definition 2. *We call two fractions i/j and i'/j complementary if $i+i'=j$.*

Summarizing these two definitions yields:

Corollary 1. *By definition two complementary fractions have the same parent.*

Theorem 1. *The numerical value of each fraction in the graph (except $1/1$) is in the open interval $(0,1)$.*

Proof. This is an immediate consequence of defining each denominator as a sum of two positive terms, one of which is the numerator. □

Corollary 2. *Fractions of the form i/i with $i>1$ don't exist in (nodes of) the graph.*

Algorithm 1. *The parent of a fraction i/j is given by the fraction $\min(a/b, b/a)$ where $a \equiv i$ and $b \equiv j-i$ [1, (21)]. Because $i \neq j$ (see above), $b>0$, so one of the two candidates a/b and b/a is smaller than 1, the other greater than one. Take the smaller to obtain the parent.*

Taking k steps exclusively down the 0-branches starting at i/j reaches the following fraction:

$$\frac{i}{j} \mapsto \frac{i}{i+j} \mapsto \frac{i}{2i+j} \mapsto \ldots \mapsto \frac{i}{ki+j}. \tag{1}$$

Remark 1. *This accumulation of multiples of the numerator in the denominator also means that walking as many 0-branches upwards as possible starting at some i/j is equivalent to the first step of the Euclid algorithm to compute (i,j), as for the Calkin-Wilf Tree [6].*

Definition 3. *$F_0=0$, $F_1=1$ and $F_i=F_{i-1}+F_{i-2}$ is the sequence of Fibonacci Numbers* [7, A000045].

Taking k steps exclusively down the 1-branches starting at i/j reaches the following nodes:

$$\frac{i}{j} \mapsto \frac{j}{i+j} \mapsto \frac{i+j}{i+2j} \mapsto \frac{i+2j}{2i+3j} \mapsto \frac{2i+3j}{3i+5j} \mapsto \ldots \mapsto \frac{F_{k-1}i+F_kj}{F_ki+F_{k+1}j}. \tag{2}$$

2. Binary Tree of Fractions

Theorem 2. *Each fraction i/j has a unique parent. So the graph has no cycles and is a tree. Leaving alone $1/1$ which has only one chile, this is a rooted binary tree with the root at $1/2$.*

Proof. The previous algorithm shows that each node has a unique parent. □

Theorem 3. *All fractions i/j in the tree are reduced: $(i,j)=1$, where $(.,.)$ denotes the greatest common divisor.*

Proof. [1] Assuming that the greatest common divisor in a node i/j is $(i,j)=d$ with some $d>1$ leads to a contradiction as follows: the generation of a unique parent of the node by keeping $a=i$ and subtracting $b=j-i$ preserves any common divisor.

[1] Not new. Appears already in Antti Karttunen's web pages of 2009.

Walking upwards through the tree generates numerators and denominators that become smaller and smaller, such that one arrives at the fraction d/d with $d > 1$. Such a fraction does not exist in the graph (following Theorem 1). □

Theorem 4. *Each denominator j occurs at $\varphi(j)$ nodes in the tree, where $\varphi(.)$ is Euler's totient function* [7, A000010].

Proof. This is a consquence of the fact that the numerators of all fractions (except 1/1) are smaller than and coprime to the denominators according to Theorem 3, and that all of these fractions are placed at one unique place in the tree with Theorem 2. □

Remark 2. *The denominator at each node is a sum of the two values of the parent node. Denominators are created by addition chains of coprime terms.*

Example 1. *The denominator of* 10/17 *is created by* $1/1 \mapsto 1/2 \mapsto 1/3 \mapsto 3/4 \mapsto 3/7 \mapsto 7/10 \mapsto 10/17$. *The addition chain is* $1+1=2$, $1+2=3$, $1+3=4$, $3+4=7$, $7+3=10$, $10+7=17$.

Definition 4. *We call the graph-theoretical distance (number of steps along the shortest path of the undirected graph) from i/j to* 1/1 *the* level *of i/j.*

So 1/2 has level 1, 1/3 and 2/3 have level 2, and so on.

Corollary 3. *There are 2^{l-1} distinct fractions at level l, $l \geq 1$.*

Proof. This is an immediate consequence of the fact that this is a binary tree and that each fraction occurs only once in the tree (using recursively the algorithm of grabbing a unique parent), and therefore cannot occur twice at the same level. □

An immediate consequence of Definition 2 is:

Corollary 4. *Two complementary fractions are at the same level.*

The fraction $1/j$ is reached by always selecting the 0-branch at all levels:

Theorem 5. *$1/j$ appears at level $j-1$.*

There is a standard mapping of the binary words of length $l-1$ to the fractions at level l: assign the empty word to 1/2. Attach the letter 0 for each step walking the 0-branch and the letter 1 for each step walking the 1-branch. $\{\} \mapsto 1/2$, $0 \mapsto 1/3$, $1 \mapsto 2/3$, $00 \mapsto 1/4$, $01 \mapsto 3/4$, $10 \mapsto 2/5$, $11 \mapsto 3/5$ and so on. A run-length encoding of such a binary word shows how many consecutive 0- or 1-branches are chosen to reach some i/j. Remark 1 vaguely associates the number of 0-runs with the number of iterations in Euclid's algorithm to compute (i, j).

3. Set of Numerators at Level l

The set of numerators at level l is the union of the set of numerators at level $l-1$ and the set of denominators at level $l-1$; The numerators copied via the 0-branches, the denominators via the 1-branches.

Definition 5. *$N(l)$ is the size of the set of numerators at level l* [7, A294443]. *$D(l)$ is the size of the set of denominators at level l* [7, A294444].

Theorem 6. *The size of the set of distinct numerators is a monotonously increasing function of the level: $N(l) \geq N(l-1)$.*

Proof. The construction preserves the numerators at level l by copying them along the 0-branch. So the set of numerators cannot decrease. □

Because complementary fractions share the same denominator, Corollary 3 leads to

Corollary 5. *There are at most 2^{l-2} distinct denominators at level l: $D(l) \leq 2^{l-1}$ for $l \geq 2$.*

So a study of the increase of the order of the numerator set is essentially a study of the first differences. The set of distinct numerators at level l grows as follows:

$$N(l) = 1, 1, 2, 3, 5, 7, 13, 20, 31, 48, 78, 118, 191 \ldots; \quad l \geq 0. \tag{3}$$

By computing the first differences, the numerators that are new in level l are [7, A295783]

$$N^+(l) \equiv N(l) - N(l-1) = 0, 1, 1, 2, 2, 6, 7, 11, 17, 30, 40, 73 \ldots; \quad l \geq 1. \tag{4}$$

The number of distinct denominators at level l grows as follows:

$$D(l) = 1, 1, 1, 2, 3, 6, 10, 16, 29, 51, 83, 148, 246, \ldots; \quad l \geq 0. \tag{5}$$

The fact that $D(l)$ grows faster than $N^+(l)$ illustrates that not all denominators at level l introduce 'new' numerators at level $l+1$.

The smallest denominator at level l (i.e., $l+1$) is obtained by walking exclusively along 0-branches. The largest denominator at level l (i.e., F_{2+l}) is obtained by walking exclusively along 1-branches [insert $i = j = 1$ and $k = l$ in (2)], which generates the fractions of ratios of consecutive Fibonacci numbers [8]. So a rough upper bound for the $D(l)$ is the length of the interval between these [7, A000126]:

$$D(l) \leq F_{2+l} - l = 1, 1, 1, 2, 4, 8, 15, 27, 47, 80, 134, \ldots l \geq 0. \tag{6}$$

Remark 3. *$F_{2+l} - l$ has ordinary generating function $(1 - 2x + 2x^3)/[(1-x)^2(1-x-x^2)]$.*

4. Entry Level

Definition 6. *The smallest level at which denominator j occurs is called the entry level of j, $E(j)$:*

$$E(j) = 0, 1, 2, 3, 3, 5, 4, 4, 5, 5, 5, 5, 5, 6, 6, 6, 6, 6, 6, 7, 6, 7, 7, 7, 7, 7, 7, 8, 7, 7, 7, 8, 8, 7 \ldots j \geq 1 \tag{7}$$

Remark 4. *The entry level is probably the same as* [7, A178047]. *This has been verified numerically for the first ten thousand values via the b-file in* [7, A178031].

An immediate (but weak) consequence of Theorem 5 is:

Theorem 7. *The entry level of j is $E(j) < j$.*

And by walking the 1-branch from there:

Theorem 8. *The numerator i occurs not later than at level i:*

$$N(l) \geq l. \tag{8}$$

Fibonacci numbers occur 'early' through the 1-branches:

$$E(F_{n+2}) = n. \tag{9}$$

For a numerical assessment of $N(l)$ one could ask for all candidates i of numerators at level l—as counted by (6)—, at which level they were introduced into the set of numerators. This implies to take a 'flight back' from i/j at that level staying as long as possible on the 0-branch to locates the fraction where it was 'lifted' from the denominator of the sum of its parent (Remark 1). This in turn asks for the shortest addition chain that could generate that denominator (Remark 2), namely the entry level.

From the point of efficiency that would actually re-do the analysis for all numerators that were already present at level $l-1$; so one could target better the computation of $N^+(l)$, the number of numerators introduced from denominators of level $l-1$.

A denominator occurs first at entry level $E(j)$, and is 'lifted' to a numerator at level $1+E(j)$, so it adds to the $N(l)$ for all $l \ge 1+E(j)$. If we count the number of occurrences of j in the sequence (7) as containing one 0, one 1, one 2, two 3's, two 4's, six 5's…:

$$E(j) \sim 0^1, 1^1, 2^1, 3^2, 4^2, 5^6, 6^7 \ldots \tag{10}$$

we see that the entry levels contain all the information to compute the first differences (4):

$$N^+(l+1) = \#\{j : E(j) = l\}. \tag{11}$$

A safe deduction of N^+ from a finite set of $E(j)$ requires that one knows an upper index j at which $E(j)$ assumes some prescribed l. The answer is given by the 'late inverse' (9):

$$E(F_{n+2}+i) > n, \quad i \ge 1. \tag{12}$$

Problem 1. *How do we compute efficiently the entry levels? For a given denominator j, which (coprime) numerator i leads to the shortest addition chain, the shortest path from the root of the tree to any i/j? Is there a pre-screening which tells us which i relate to shorter addition chains? Can we know beforehand that* $3/11$*, (complementary to* $8/11$*) and* $4/11$ *(complementary* $7/11$*) have lower level than* $5/11$ *(complimentary* $6/11$*), and* $2/11$ *(complementary* $9/11$*)?*

With Corollary 4, the size of the set of candidates i is effectively $\varphi(j)/2$, $j > 2$, because only one of the two numerators of complimentary fractions needs to be investigated.

Algorithm 2. *To compute $E(j)$, consider all i/j where $(i,j) = 1$, $1 \le i < j/2$, compute their level by measuring the number of reduction steps in Algorithm 1, then take the smallest of these step numbers.*

Starting from $i = 1$ is probably a bad choice, because $1/j$ needs to take the long path with $j-1$ steps back through the 0-branches to reach the root node $1/2$, see Theorem 7. This leads to:

Algorithm 3. *To compute $E(j)$, take $E(j) = j-1$ as an upper bound, then consider all i/j where $(i,j) = 1$, $2 \le i < j/2$, compute their level by measuring the number of reduction steps in Algorithm 1. $E(j)$ is the smallest of these step numbers.*

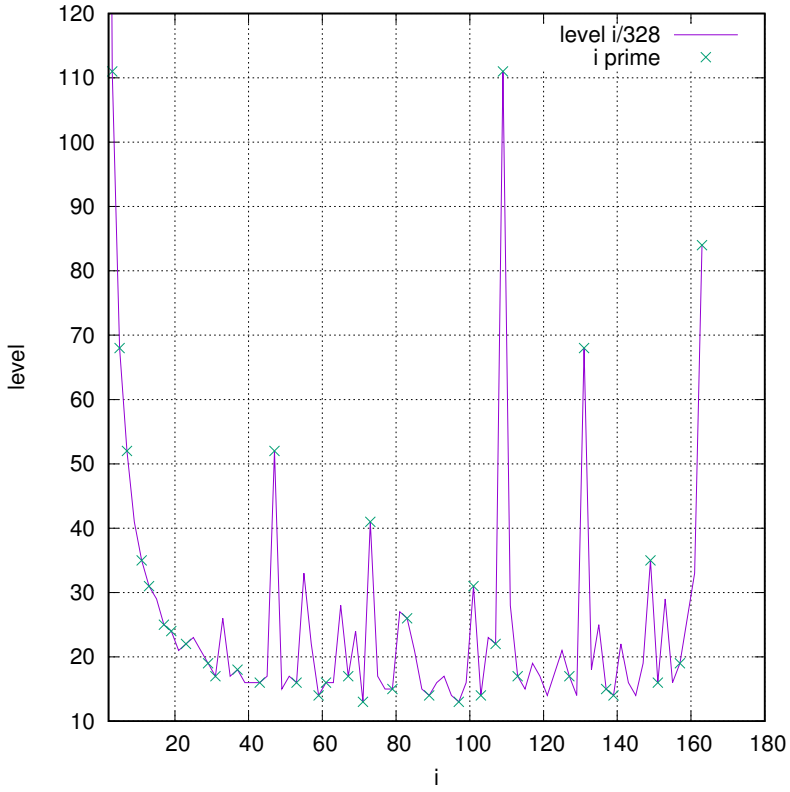


FIGURE 1. Examples of the levels of the fractions $i/328$ for $(i, 328) = 1$ in the relevant search range $2 \leq i \leq 328/2$. The values where i is a prime number are marked with crosses. The smallest level is $E(328) = 13$, reached for $i = 71$ (word 010111111000) and $i = 97$ (word 001111110100).

The estimation of a numerator i such that i/j has small level is not easy, as demonstrated in Figure 1 and 2.

5. JAVA IMPLEMENTATION

The source code of the file `KepTreeNode.java` follows.

```
import java.util.* ;
import java.lang.* ;
import java.math.* ;
/*!***************************************************
* @brief A reduced fraction num/den in the range 0< num/den <= 1.
* @autor R. J. Mathar
* @since 2017-12-01
*/
class KepTreeNode
{
    /** The numerator
    */
```

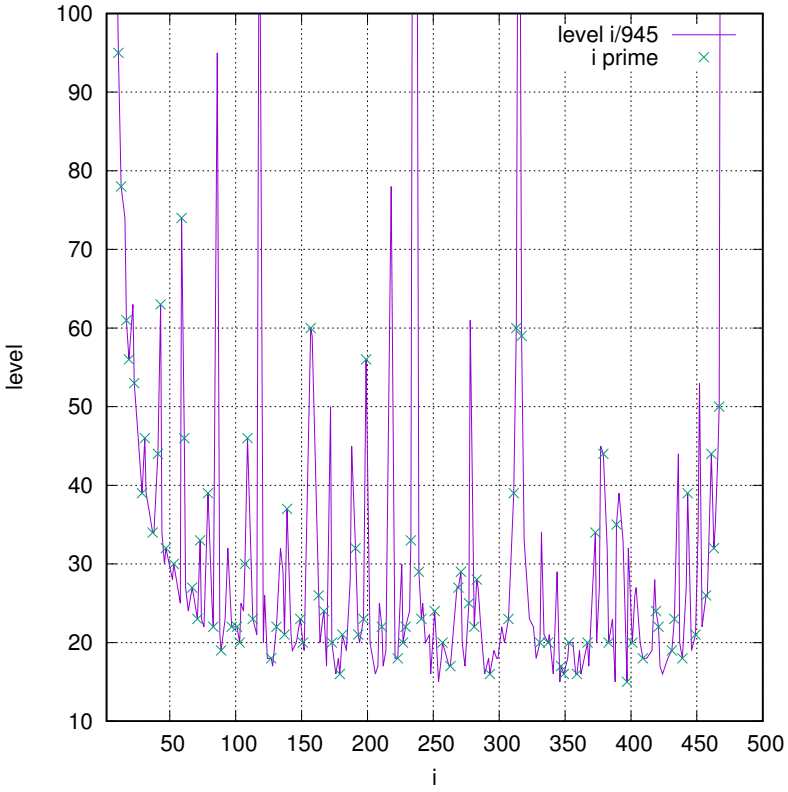


FIGURE 2. Examples of the levels of the fractions $i/945$ for $(i, 945) = 1$ in the relevant search range $2 \le i \le 945/2$. The values where i is a prime number are marked with crosses. The smallest level is $E(945) = 15$, reached for $i = 254$ (word 11011011101100), 346 (word 01101110110110), 388 (word 10111101001010) and 397 (word 10100101111010).

```
    BigInteger num ;
    /** The denominator
    */
    BigInteger den ;

    static BigInteger two = BigInteger.valueOf(2) ;

    /** Static list of Fibonacci numbers.
    * The list is long enough for most practical purposes of
    * computing the number of distinct numerators at levels.
    */
    static long[] F = {
0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987, 1597, 2584,
4181, 6765, 10946, 17711, 28657, 46368, 75025, 121393, 196418, 317811, 514229,
832040, 1346269, 2178309, 3524578, 5702887, 9227465, 14930352, 24157817,
39088169, 63245986, 102334155, 165580141, 267914296, 433494437, 701408733L,
```

```
1134903170L, 1836311903L, 2971215073L, 4807526976L, 7778742049L, 12586269025L,
20365011074L, 32951280099L, 53316291173L, 86267571272L, 139583862445L,
225851433717L, 365435296162L, 591286729879L, 956722026041L, 1548008755920L};

    /*!*********************************
    * @param a Then numerator
    * @param b Then denominator
    * @throws ArithmeticException
    */
    KepTreeNode(BigInteger a, BigInteger b)
    {
        num=a ;
        den=b ;
        if ( num.compareTo(den) > 0 )
            throw new ArithmeticException(num.toString() + " not smaller " + den.toString()) ;
        if ( num.gcd(den).compareTo(BigInteger.ONE) > 0 )
            throw new ArithmeticException(num.toString() + " not copr to " + den.toString()) ;
    }

    /*!*********************************
    * @brief represent the fraction as the usual slanted fraction.
    * @return A string representation i/j.
    */
    public String toString()
    {
        return new String(""+num+"/"+den) ;
    }

    /*!****************************
    * @brief construct the parent node.
    * @return the node that is one level higher up
    */
    KepTreeNode parent()
    {
        BigInteger tmp = den.subtract(num) ;
        if ( tmp.compareTo(num) < 0 )
            return new KepTreeNode(tmp,num) ;
        else
            return new KepTreeNode(num,tmp) ;
    }

    /*!****************************
    * @return the binary word representing this fraction.
    */
    public String binWord()
    {
        String w =new String();
        if ( den.compareTo(two) > 0)
        {
            BigInteger tmp = den.subtract(num) ;
            if ( tmp.compareTo(num) < 0 )
            {
                /* took 1 branch */
```

```
                return parent().binWord() + "1" ;
            }
            else
            {
                /* took 0 branch */
                return parent().binWord() + "0" ;
            }
        }
        return w;
    }


    /*!************************
    * @brief determine the level in the Kepler tree of fractions
    *  Recursive algorithm which runs upwards in the tree up to 1/2 and 1/1.
    *  This is overall the simplest algorithm, and is in practise
    *  replaced by leel(long mxlvl).
    * @return a number >=0 where 1/1 is at level 0.
    */
    int level()
    {
        if ( den.compareTo(BigInteger.ONE) == 0 )
            return 0 ;
        else
        {
            return 1+parent().level() ;
        }
    } /* level */

    /*!************************
    * @brief determine the level in the Kepler tree of fractions if smaller than mxlvl
    *  Recursive algorithm which runs upwards in the tree up to 1/2 and 1/1.
    * @param mxlvl A cap on the search depth. Return levels accurately
    *   only if smaller than mxlvl, but report larger levels just as mxlvl.
    * @return a number >=0 where 1/1 is at level 0.
    */
    int level(long mxlvl)
    {
        if ( den.compareTo(BigInteger.ONE) == 0 )
            return 0 ;
        else if ( den.compareTo(two) == 0 )
            return 1 ;
        else if ( mxlvl <= 0)
            return 0 ;
        else
        {
            int l=0 ;
            BigInteger a = num ;
            BigInteger b = den ;
            do
            {
                /* figure out the number of steps one could
                * run along 0-branches (Euclid's algorithm)
```

```
            */
            final BigInteger k = b.divide(a).subtract(BigInteger.ONE) ;
            if (k.compareTo(BigInteger.ZERO) > 0 )
            {
                b = b.subtract( a.multiply(k) ) ;
                l += k.longValue() ;
            }
            if ( b.compareTo(BigInteger.ONE) == 0 )
                break ;
            final BigInteger tmp = b.subtract(a) ;
            /* because we have exhausted the number of
            * 0-steps above, we know that tmp < a now.
            * So the next step is along a 1-branch.
            */
            {
                /* parent is tmp/a */
                b =a ;
                a = tmp ;
            }
            l++ ;
            if ( b.compareTo(BigInteger.ONE) == 0 )
                break ;
        } while( l < mxlvl) ;
        return l ;
    }
} /* level */

/*!**************************
* @return true if i is a prime number or 1
*/
static boolean isprime(long i)
{
    if ( i == 1L)
        return true;
    for(long d = 2L ; d*d <= i; d++)
        if ( i % d == 0 )
            return false;
    return true ;
}

/*!**************************
* @brief compute the smallest level at which denominator j appears.
* @param j representative denominator
* @param verb if true print the levels for all relevant i
* @param iprime if true print the levels only for prime i.
* @return the smallest level of all admitted i/j.
*/
static long entryLevel(long j, boolean verb, boolean iprime)
{

    long l=j-1 ;
    BigInteger b = BigInteger.valueOf(j) ;
    /* test coprime values up to j/2
```

```
        */
        if ( false )
        {
            /* explicit but slow version which computes
            * all levels for all coprime (i,j)
            */
            for(long i = 1 ; 2*i < j ; i++)
            {
                BigInteger a = BigInteger.valueOf(i) ;

                if ( a.gcd(b).compareTo(BigInteger.ONE) == 0 )
                {
                    KepTreeNode ij = new KepTreeNode(a,b) ;
                    int thisl = ij.level() ;
                    if ( verb)
                        if ( !iprime || isprime(i) )
                            System.out.println(i + " " + thisl + " " + ij.binWord()) ;
                    if ( thisl < l)
                        l = thisl ;
                }
            }
        }
        else
        {
            if ( j == 1L )
                return 0 ;
            else if ( j == 2L )
                return 1 ;

            /* faster version which prunes the walks upwards
            * for numerators knowing which upper limits have
            * been set by earlier numerators.
            */
            for(long i = j/2 ; i > 1 ; i--)
            {
                final BigInteger a = BigInteger.valueOf(i) ;

                /* search limited to coprime numerators
                */
                if ( a.gcd(b).compareTo(BigInteger.ONE) == 0 )
                {
                    KepTreeNode ij = new KepTreeNode(a,b) ;
                    int thisl = ij.level(l) ;
                    if ( thisl < l)
                        l = thisl ;
                }
            }
        }
        return l ;
    }

    /** Frequencies of the N^+ up to some maximum index
    * @return [i] is the number of numerators new in entry level i.
```

```
*/
static int[] Nplus(int maxi)
{
    int[] npl = new int[1+maxi] ;
    npl[0] = 1 ;
    npl[1] = 0 ;
    /* to gather results for new numerators up to level maxi, need to
    * compute entry levels for denominators up to  level maxi-1,
    * up to fibonacci(2+maxi)
    */
    long highj = F[1+maxi] ;
    int fidx =3 ;
    for(int j=2 ; j <= highj ; j++)
    {
        long l = entryLevel(j,false,false) ;
        if ( l+1 < npl.length)
        {
            npl[(int)(l+1)]++ ;
        }

        /* print intermediate results as soon as the denominator
        * has passed a fibonacci number, and npl[] will not
        * increase further.
        */
        if ( j == F[1+fidx] )
        {
            System.out.println(npl[fidx-1]) ;
            fidx++ ;
        }
    }
    return npl ;
}

/*!*********************************
* To print a list of N+[] up to some maximum denominator:
*  javac -cp . KepTreeNode.java
*  java -cp . KepTreeNode <maxdenominator>
*/
static public void main(String[] args)
{




    /* b-file A178047, add 1 for A178031
    for(int j=1 ; j < 10000 ; j++)
        System.out.println(j + " " + (1+KepTreeNode.entryLevel(j,false,false)) ) ;
    */
```

```
        int maxi = Integer.valueOf(args[0]).intValue() ;
        int[] firsti = Nplus(maxi) ;
        for(int i=0 ; i < firsti.length ; i++)
            System.out.println(i+ " " + firsti[i]) ;

        /* print also the partial sums
        */
        int psum = 0;
        for(int i=0 ; i < firsti.length ; i++)
        {
            psum += firsti[i] ;
            System.out.println(i+ " " + psum) ;
        }

    }

} /* KepTreeNode */
```

References

1. Marco Abrate, Stefano Barbero, Umberto Cerruti, and Nadir Murru, *Construction and composition of rooted trees via descent functions*, Algebra **2013** (2013), 543913.
2. J. Bruce Brackenridge, *Kepler, elliptical orbits, and celestial circularity: A study in the persistence of metaphysical commitment*, Ann. Sci. **39** (1982), no. 2, 117–143. MR 0677680
3. Michel Dekking, *Substitution invariant sturmian words and binary trees*, arxiv:1705.08607 (2017).
4. Stephen Alan Eberhart, *Classical quadirivum and kepler's hormonice mundi*, Master's thesis, Univ. of Montana, 1982.
5. Clark Kimberling and Peter J. C. Moses, *The infinite fibonacci tree and other trees generated by rules*, Fib. Quart. **52** (2014), no. 5, 136. MR 3479495
6. Aimeric Malter, Dierk Schleicher, and Don Zagier, *New looks at old number theory*, Am.. Math. Monthly **120** (2013), no. 3, 243–264.
7. Neil J. A. Sloane, *The On-Line Encyclopedia Of Integer Sequences*, Notices Am. Math. Soc. **50** (2003), no. 8, 912–915, http://oeis.org/. MR 1992789
8. Lawrence Smolinsky, *Features of a high school olympiad problem*, arXiv:1602.08028 (2016).

Max-Planck Institute of Astronomy, Königstuhl 17, 69117 Heidelberg, Germany