Labeled Chip Firing on Binary Trees

Monique Roman and Jasper Hugunin

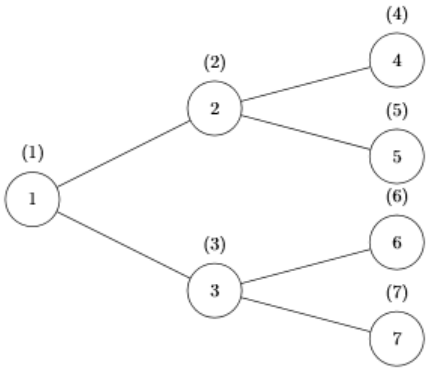
August 9, 2017

1 Introduction

The chip firing game has its roots in the abelian sandpile model that was first introduced by Bak, Tang and Wiesenfeld and later refined by Dhar. [1][2] The labeled chip firing process created by Hopkins, McConville, and Propp, investigates chip firing on the infinite path graph $\mathbb{Z}$. Consider this: "we start with n labeled chips $(1), (2), ..., (n)$ at the origin; at each step we choose any two chips (a) and (b) with $a < b$ that occupy the same vertex i and fire these chips together, moving (a) to vertex $i-1$ and (b) to vertex $i+1$; we keep carrying out firings until no chips can fire." [3] This will serve as the motivation for the chip firing process on binary trees. We will first specify the rules for labeled chip firing on trees. Then we will discuss the construction of the algorithm where a strict sorting is achieved. A python program will simulate the algorithm. Finally, we will look at a reverse algorithm that will also strictly sort the chips.

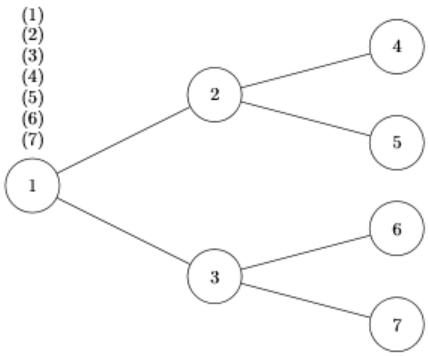
2 Modification of Labeled Chip Firing Process for Trees

We will be looking at undirected binary trees, where the vertices are labeled using the positive integers and the root is oriented on the leftmost side. The objective of the modified chip firing process is to have a final configuration that is sorted such that all the chips on $l-1$ are less than all the chips on l, i.e. $l > (l-1) > (l-2)... > (l-n)$ such that when $l = n$ will represent the root. We also want chips to land on the vertex of same corresponding value, meaning (8) would end up on vertex 8 in the final configuration. This is what is defined as strictly sorted, as pictured below.

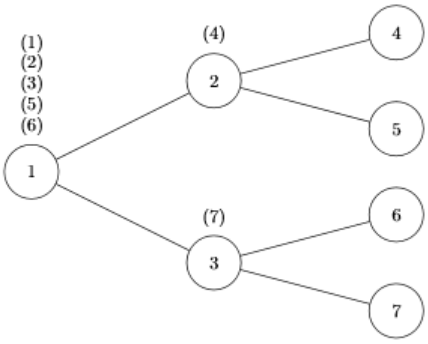


We will start with as many chips as vertices on the graph (i.e. $n = |V|$ placed at the root. The set of edges (i.e. $E = (1,2), (1,3), (2,4), (2,5), (3,6), (3,7)...$) will have a local ordering such that for any given pair of edges of the type (d,e) and (d,f) will imply $(d,e) < (d,f)$. In other words, edge $(1,2)$ will have a lower value than the edge $(1,3)$. Meaning, if $(a) < (b)$, (a) would fire along edge $(1,2)$ and (b) would fire along edge $(1,3)$. As per convention, we can fire as many chips as a vertex's out degree.

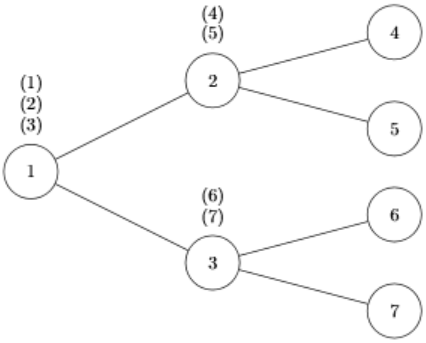
Let us consider the binary tree of level 2. Below will be the list of steps that show the sequences of firing along with accompanying pictures.



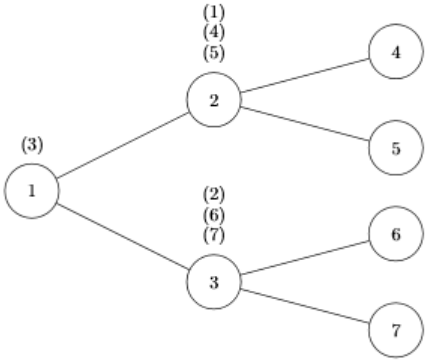
Initial Configuration of n=7



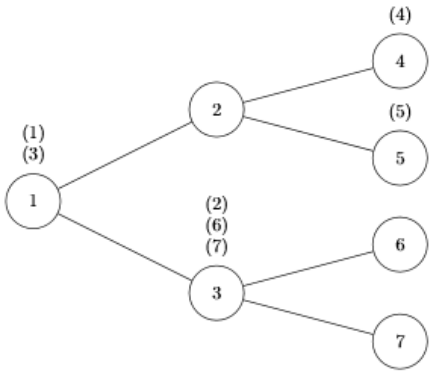
Step 1: Fire chips (4) and (7)



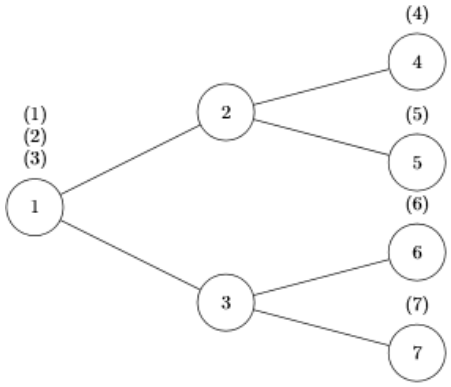
Step 2: Fire chips (5) and (6)



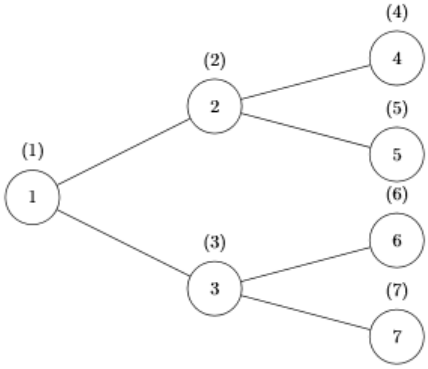
Step 3: Fire chips (1) and (2)



Step 4: Fire chips (1), (4), and (5)



Step 5: Fire chips (2), (6), (7)



Step 6: Fire chips (2) and (3)

As shown above, when dealing with firings of three chips, we will fire the chip that has the lowest value of the three chip down one level. The chip that is of middle value will be fired along the upper branch on the next level. The chip that is of the highest value of the three will fire along the lower branch on the next level. Even though the root is unstable, once a vertex has reached its out degree, it will take precedent over the root during the firing process. This will only happen once all vertices within a certain level have reached their out degree, the firing at one particular vertex at such a level will force the rest of the chips within that level to be fired at their respective vertices. This is illustrated by steps 4 and 5. Also, lower and upper branches will be defined as the the edges that are incident to the chip where the firing takes place and are derived from the ordering on the edges. In the algorithm, we will only consider the chip firings from the origin, which we will call quads, not the clean up moves that occur past $l = 1$. A quad is a sequence of two pairs of chips being fired from the root, as shown in steps 1 and 2. Clean up moves come in two forms:place holder moves and sweeper moves. A place holder move directly follows a quad and will be a firing of (1) and (2) from the root with the intent to push higher valued chips outward, shown in step 3. A sweeper move is a sequence moves that happens once all the vertices within a level reach their respective out degree and at least one firing happens, forcing all other vertices within that level to fire, again illustrated by steps 4 and 5.

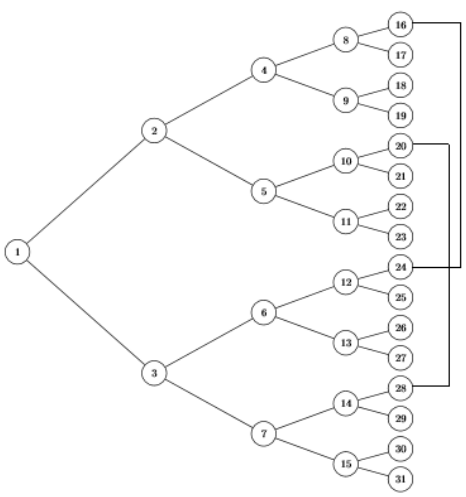
Claim 1. If you follow those rules, then a firing sequence will be uniquely determined by how the vertices are fired from the origin.

Thus, we'll specify in our algorithm in the next section by saying how the vertices are fired from the origin by use of quads.

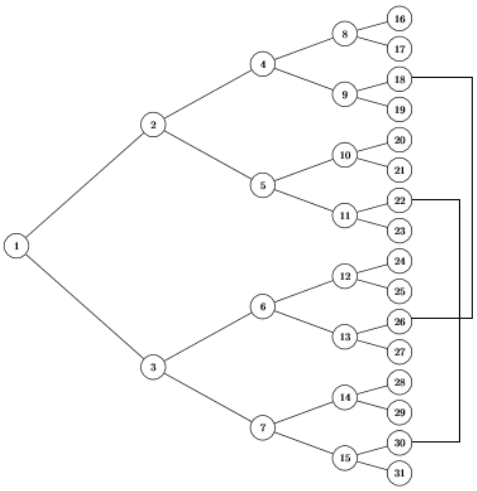
2.1 Refined Algorithm

The algorithm will be take into consideration the chip firing processes presented in section 2.4 by dividing major parts of the processes into quads. It will not, however, consider the clean up moves

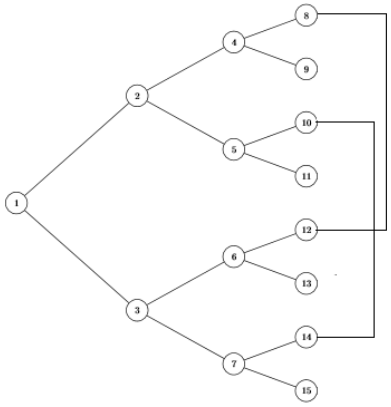
We will graphically show the quads of a tree of level 4 below:



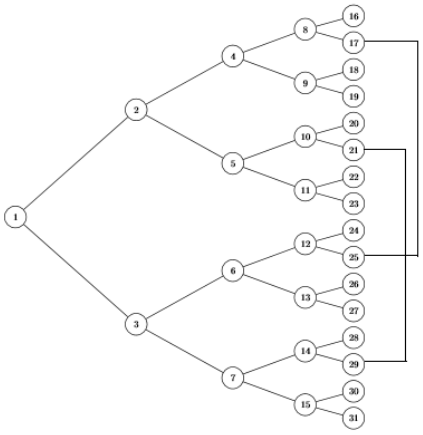
l_1



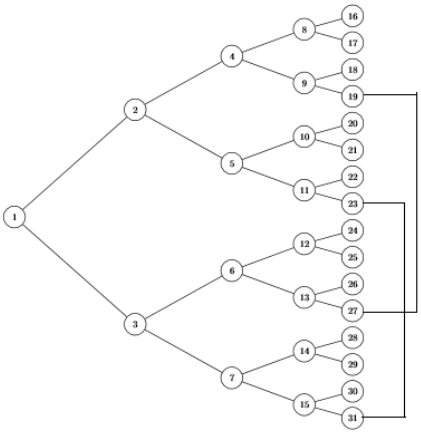
l_2



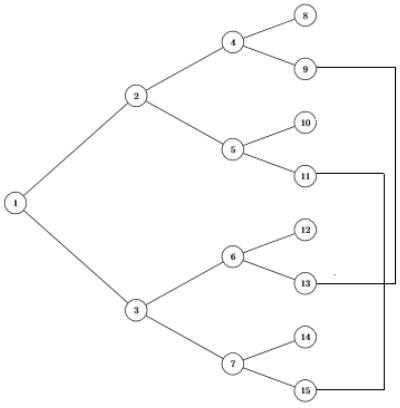
$(l-1)_1$



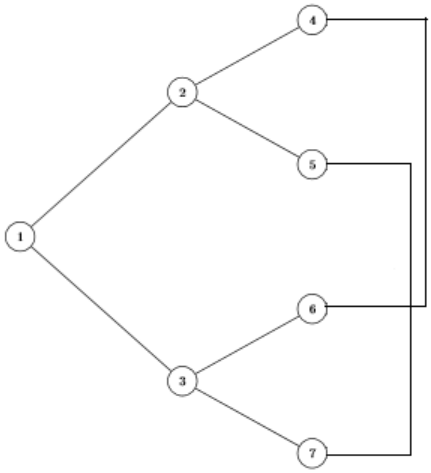
l_3



l_4



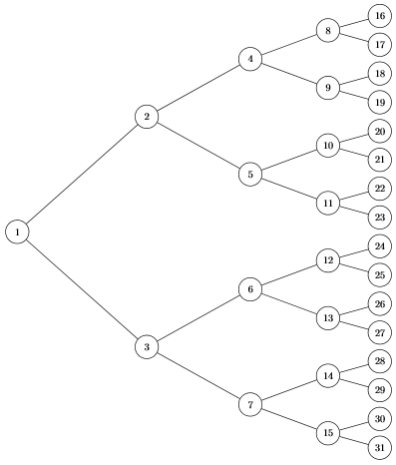
$(l-1)_2$



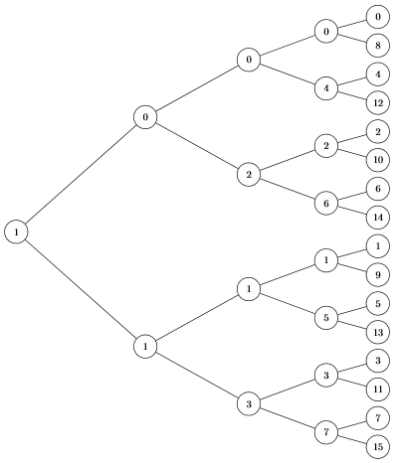
$(l-2)_1$

The question now is how to strategically choose the quads. We can transform the original binary tree we have to another binary tree that has the vertices labeled in a way that will help us create the quads. Below will have one such transformation for a binary tree of level 4:

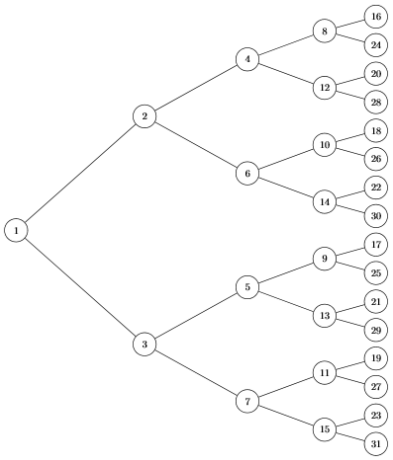
As shown below in the intermediate mapping, within a given level, the vertices are labeled starting from 0 to $2^l - 1$. The interesting thing about this mapping is that the vertices will follow reverse binary incremeneting. We will illustrate this process below with level 4



Original Binary Tree



Intermediate Mapping



Final Mapping

Vertex	Associating Binary Number of Vertex	Removal of First Bit	Reverse of New Bit	Associated Decimal Number of Reversed Bit	φ of Vertex
16	10000	0000	0000	0	16
17	10001	0001	1000	8	24
18	10010	0010	0100	4	20
19	10011	0011	1100	12	28
20	10100	0100	0010	2	18
21	10101	0101	1010	10	26
22	10110	0110	0110	6	22
23	10111	0111	1110	14	30
24	11000	1000	0001	1	17
25	11001	1001	1001	9	25
26	11010	1010	0101	5	21
27	11011	1011	1101	13	29
28	11100	1100	0011	3	19
29	11101	1101	1011	11	27
30	11110	1110	0111	7	23
31	11111	1111	1111	15	31

For the last column. φ is defined by these equations:

$$x = 2^l + \sum_{i=0}^{l-1}(\alpha_i)(2^i) \qquad \alpha_i \in \{0,1\}$$

$$\varphi(x) = 2^l + \sum_{i=0}^{l-1}(\alpha_{l-1-i})(2^i)$$

We can now use φ in order to specify which chips are in each quad: $l_{i+1} = \varphi(4i + j + 2^l)$, such that $j \in \{0,1,2,3\}$.

Below will be the quads of sections within the chip firing process up to level 5:
Level 2:

1. l_1

Level 3:

1. l_1
2. l_2
3. $(l-1)_1$

Level 4:

1. l_1
2. l_2
3. $(l-1)_1$
4. l_3
5. l_4
6. $(l-1)_2$
7. $(l-2)_1$

Level 5:

1. l_1
2. l_2
3. $(l-1)_1$
4. l_3
5. l_4
6. $(l-1)_2$
7. $(l-2)_1$
8. l_5
9. l_6
10. l_7
11. l_8
12. $(l-1)_3$
13. $(l-1)_4$
14. $(l-2)$
15. $(l-3)$

The pattern for the quads is not clear at first glance. However, upon closer investigation, we see that the first seven quads of Level 5 are the exact seven quads found for Level 4. But then how are the other eight quads within a level 5 found? They are essentially quads that will push all the remaining chips on level l, then $(l-1)$, and so forth until the last quad at level 2 takes place.

In essence, the algorithm that is derived from the quads of a particular binary tree of level, l, is as follows:

1. The first less than half of the quads will come from $(l-1)$
2. The remaining quads will try to push out the farthest chips that will end up on l, then $(l-1)$, until the last quad at level 2 takes places.

Due to the rules we placed on the chip firing process, we resolved that the only important part in constructing the algorithm is figuring out how to strategically choose chips prior to all (4)...(n) being fired. This leads to the following theorem.

Theorem 1. *Suppose we fire the chips according to the algorithm, once (4)...(n) have been fired, the chips will topple from that point onward until the strict sorting is achieved.*

(Note: Theorem is proved in section 2.4)

2.2 Alternative Reverse Algorithm

In an opposite approach from the previous algorithm, this algorithm fires low numbered chips first. This is `algorithm2` in the program below, and has also been tested up to level 10.

First, we fire chips 2 and 3. This fills out level 1. From now on, whenever chips 2 and 3 return to the root (because level 1 was fired), we again fire them from the root. We then fire the chips in deeper levels. The i-th pair of chips (starting from 0) that we fire in level l are computed by taking the reversal of the binary bits of $2i$ and $2i+1$ viewed as a word of l bits, and adding 2^l (so that we are offsetting into the chips from level l). This ordering is chosen so that we never have two chips on one node that need to go in the same direction.

This algorithm has the pleasant property that it passes through solved configurations for trees of smaller depth; that is, it transforms a tree with chips 1 to $2^l - 1$ on their respective nodes into a tree with chips 1 to $2^{l+1} - 1$ on their respective nodes.

We can attempt to generalize this to trees where each node on the same level has the same branching factor. Let the branching factor of level l be b_l. Let there be n levels. Label the nodes and chips by their path, $i = i_0 i_1 \dots i_{l-1}$, and $0 \le i_l < b_l$. The firing will be done analogously to the binary tree case, where we order chips by first length and then lexicographical order, and send the smallest one towards the root, and the rest down the tree. We fire the chips in a different order. Start with $t = t_1 t_2 \dots t_k$ being the empty string (corresponding to the root), and repeatedly increment t_1, carrying to the right when $t_l = b_{l+1}$ to enumerate a sequence of t. Fire chips $0t, 1t, \dots, (b_0 - 1)t$ from the root, and then increment t. Whenever level 1 fires, sending chips (which will be chips $0, 1, \dots, b_0 - 1$) back to the root, immediately fire again from the root (same as the binary tree case).

We claim that before firing at state t, the chips on the tree are: all yet unfired chips at the root (which come later in the enumeration than $0t, 1t, \dots (b_0 - 1)t$), if $|i| > |t|$, then no chips on node i, and otherwise, with $|i| = l$ and $|t| = k$, we have chip i and chips $i_0 i_1 \dots i_{l-1} j t_{l+1} t_{l+2} \dots t_k$ where $j \in \{0, 1, \dots, t_l - 1\}$.

2.3 Program based off of Algorithm

Below is a program that was created from the algorithm and is valid for binary trees up to level 10.

```
class TreeModel():
    def __init__(self, level):
        self.level = level
        self.numnodes = 2 ** (level + 1) - 1
        self.model = [set() for i in range(self.numnodes)]
        self.model[0] = set(range(1, self.numnodes + 1)) # one-based

    def is_stable(self):
        return all(len(node) < 3 for node in self.model[1:])

    def fire(self, x1, x2):
        """Fire chips x1, x2 from the root, and stabilize."""
```

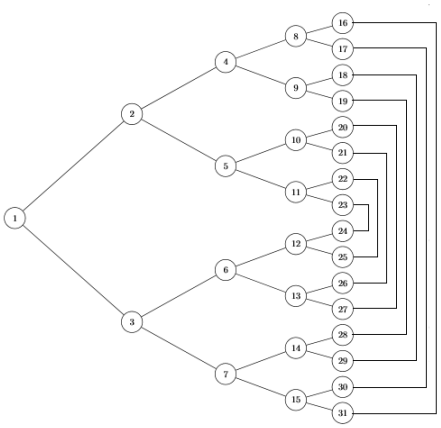
```
        if not (x1 in self.model[0] and
                x2 in self.model[0] and
                x1 != x2):
            raise ValueError("Can't fire those chips""")
        x1, x2 = min(x1, x2), max(x1, x2)
        assert x1 < x2
        self.model[0].remove(x1)
        self.model[0].remove(x2)
        self.model[1].add(x1)
        self.model[2].add(x2)
        self._stabilize()

    def _stabilize(self):
        """Stabilize the tree, called internally after each fire."""
        assert self._each_level_even()
        levels = [len(self.model[2 ** level - 1])
                  for level in range(1, self.level + 1)]
        assert all(0 <= level <= 3 for level in levels)
        assert all(level < 3 for level in levels[1:])
        if levels[0] < 3: return # nothing to do
        # Under these preconditions, there is only one stable configuration
        for i in range(1, self.numnodes):
            assert len(self.model[i]) <= 3
            if len(self.model[i]) == 3:
                self._topple(i)
        assert self._each_level_even()
        assert all(len(self.model[i]) < 3 for i in range(1, self.numnodes))

    def _topple(self, i):
        """Topple node i (0-based)."""
        assert i >= 1
        assert len(self.model[i]) == 3
        x0, x1, x2 = sorted(self.model[i])
        self.model[i].clear()
        self.model[(i - 1) // 2].add(x0)
        self.model[i * 2 + 1].add(x1)
        self.model[i * 2 + 2].add(x2)

    def _each_level_even(self):
        return all(len(set(len(self.model[i - 1])
                           for i in range(2 ** level,
                                          2 ** (level + 1))))
                   == 1
                   for level in range(1, self.level + 1))

# An algorithm is an iterator object that yields pairs of chips to fire

def algorithm1(n):
    T = TreeModel(n)
    def quad(level, i):
        # yields the moves for the ith quad of level (0-based)
        assert level >= 2
        base = 2 ** level
        quart = 2 ** (level - 2)
```

```
            assert 0 <= i < quart

            # Compute the bitwise reversal of i in the size of quart
            j = 0
            for biti in range(level - 2):
                j = j * 2 + i % 2
                i //= 2
            assert i == 0

            yield (base + 0 * quart + j, base + 2 * quart + j)
            T.fire(base + 0 * quart + j, base + 2 * quart + j)
            yield (base + 1 * quart + j, base + 3 * quart + j)
            T.fire(base + 1 * quart + j, base + 3 * quart + j)
            while T.model[1]:
                T.fire(1, 2)
                yield (1, 2)

    if n == 0:
        return
    for level in range(2, n + 1):
        for i in range(level - 2):
            num = 2 ** (level - 3 - i)
            for j in range(num, 2 * num):
                yield from quad(n - i, j)
        yield from quad(n - (level - 2), 0)
    yield (2, 3)

def algorithm2(n):
    def quarto(level, i):
        """Fires the ith quarto at level."""
        # restores (2, 3) at end
        assert level >= 1
        base = 2 ** level
        quart = 2 ** (level - 2)
        assert 0 <= i < quart

        # compute bitwise reversal of i in size of quart
        itmp = i
        j = 0
        for _ in range(level - 2):
            j = j * 2 + itmp % 2
            itmp //= 2
        assert itmp == 0

        x0, x1 = base + 0 * quart + j, base + 2 * quart + j
        x2, x3 = base + 1 * quart + j, base + 3 * quart + j

        yield (x0, x1)
        yield (x2, x3)

        itmp = i
        n = 0
        # count the number of trailing 1s in i
        while itmp % 2 == 1:
```

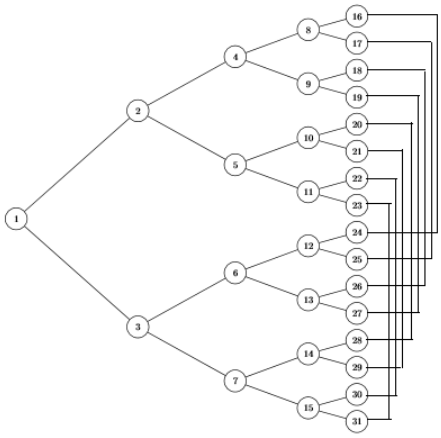
```
            n += 1
            itmp //= 2

        # This is how many times we have to fire (2, 3)
        # to get them to stick (not collapse back to the root)
        for _ in range(n + 1):
            yield (2, 3)

    yield (2, 3)

    for level in range(2, n + 1):
        for i in range(2 ** (level - 2)):
            yield from quarto(level, i)

def check(n, alg):
    T = TreeModel(n)
    for move in alg(n):
        T.fire(*move)
    accurate = all(x == {i + 1} for i, x in enumerate(T.model))
    print(f"Accurate ({n}): {accurate}")

print("Algorithm 1:")
for n in range(2, 11):
    check(n, algorithm1)

print("Algorithm 2:")
for n in range(2, 11):
    check(n, algorithm2)

##l = []
##def bump():
##    def inc(i, x=1):
##        if i < 0: pass
##        elif i < len(l): l[i] += x
##        elif i == len(l): l.append(x)
##        else: assert False
##    inc(0)
##    for i in range(len(l)):
##        if l[i] == 3:
##            l[i] = 0
##            inc(i - 1, 2)
##            inc(i + 1)
##for i in range(200):
##    bump()
##    print(' '.join(str(x) for x in l))
##
```

2.4 Chip Firing Processes for Binary Trees up to Level Four

This section provides the sequences of chip firing moves for binary trees up to level four. Each tree with level greater than 2 will have two different sequences in accordance to the pairing conventions which are specified below.



First way of pairing chips



Second way of pairing chips

Level 2

1. (4),(7)
2. (5),(6)
3. (1),(2)
4. (1),(4),(5)
5. (2),(6),(7)
6. (2),(3)

Level 2

1. (4),(6)
2. (5),(7)
3. (1),(2)
4. (1),(4),(5)
5. (2),(6),(7)
6. (2),(3)

(Note: Steps 1 and 2 are interchangeable. Similarly steps 4 and 5 are as well.)
(Note: (2), (3) will be the last move in every firing process)

Level 3

1. (8),(15)
2. (10),(13)
3. (1),(2)
4. (1),(8),(10)
5. (9),(14)

6. (11),(12)
7. (1),(2)
8. (1),(9),(11)
9. (2),(12),(14)
10. (4),(7)
11. (5),(6)
12. (1),(2)
13. (1),(4),(5)
14. (2),(6),(7)
15. (4),(8),(9)
16. (5),(10),(11)
17. (6),(12),(13)
18. (7),(14),(15)
19. (1),(2)
20. (1),(4),(5)
21. (2),(6),(7)
22. (2),(3)

Level 3

1. (8),(12)
2. (10),(14)
3. (1),(2)
4. (1),(8),(10)
5. (2),(12),(14)
6. (9),(13)
7. (11),(15)
8. (1),(2)
9. (1),(9),(11)
10. (2),(13),(15)
11. (4),(6)
12. (5),(7)
13. (1),(2)
14. (1),(4),(5)
15. (2),(6),(7)

16. (4),(8),(9)
17. (5),(10),(11)
18. (6),(12),(13)
19. (7),(14),(15)
20. (1),(2)
21. (1),(4),(5)
22. (2),(6),(7)
23. (2),(3)

Level 4

1. (16),(31)
2. (20),(22)
3. (1),(2)
4. (1),(16),(20)
5. (2),(27),(31)
6. (18),(29)
7. (22),(25)
8. (1),(2)
9. (1),(18),(22)
10. (2),(25),(29)
11. (8),(15)
12. (10),(13)
13. (1),(2)
14. (1),(8),(10)
15. (2),(13),(15)
16. (8),(16),(18)
17. (10),(20),(22)
18. (13),(25),(27)
19. (15),(29),(31)
20. (1),(2)
21. (1),(8),(10)
22. (2),(13),(15)
23. (17),(30)
24. (21),(26)

25. (1),(2)
26. (1),(17),(21)
27. (2),(26),(30)
28. (19),(28)
29. (23),(24)
30. (1),(2)
31. (1),(19),(23)
32. (2),(24),(28)
33. (8),(17),(19)
34. (10),(21),(23)
35. (13),(24),(26)
36. (15),(28),(30)
37. (1),(2)
38. (1),(8),(10)
39. (2),(13),(15)
40. (9),(14)
41. (11),(12)
42. (1),(2)
43. (1),(9),(11)
44. (2),(12),(14)
45. (4),(7)
46. (5),(6)
47. (1),(2)
48. (1),(4),(5)
49. (2),(6),(7)
50. (4),(8),(9)
51. (5),(10),(11)
52. (6),(12),(13)
53. (7),(14),(15)
54. (8),(16),(17)
55. (9),(18),(19)
56. (10),(20),(21)
57. (11),(22),(23)

58. (12),(24),(25)
59. (13),(26),(27)
60. (14),(28),(29)
61. (15),(30),(31)
62. (1),(2)
63. (1),(4),(5)
64. (2),(6),(7)
65. (4),(8),(9)
66. (5),(10),(11)
67. (6),(12),(13)
68. (7),(14),(15)
69. (1),(2)
70. (1),(4),(5)
71. (2),(6),(7)
72. (2),(3)

Level 4

1. (16),(24)
2. (20),(28)
3. (1),(2)
4. (1),(16),(20)
5. (2),(24),(28)
6. (18),(26)
7. (22),(30)
8. (1),(2)
9. (1),(18),(22)
10. (2),(26),(30)
11. (8),(12)
12. (10),(14)
13. (1),(2)
14. (1),(8),(10)
15. (2),(12),(14)
16. (8),(16),(18)
17. (10),(20),(22)

18. (12),(24),(26)
19. (14),(28),(30)
20. (1),(2)
21. (1),(8),(10)
22. (2),(12),(14)
23. (17),(25)
24. (21),(29)
25. (1),(2)
26. (1),(17),(21)
27. (2),(25),(29)
28. (19),(27)
29. (23),(31)
30. (1),(2)
31. (1),(19),(23)
32. (2),(27),(31)
33. (8),(17),(19)
34. (10),(21),(23)
35. (12),(25),(27)
36. (14),(29),(31)
37. (1),(2)
38. (1),(8),(10)
39. (2),(12),(14)
40. (9),(13)
41. (11),(15)
42. (1),(2)
43. (1),(9),(11)
44. (2),(13),(15)
45. (4),(6)
46. (5),(7)
47. (1),(2)
48. (1),(4),(5)
49. (2),(6),(7)
50. (4),(8),(9)

51. (5),(10),(11)
52. (6),(12),(13)
53. (7),(14),(15)
54. (8),(16),(17)
55. (9),(18),(19)
56. (10),(20),(21)
57. (11),(22),(23)
58. (12),(24),(25)
59. (13),(26),(27)
60. (14),(28),(29)
61. (15),(30),(31)
62. (1),(2)
63. (1),(4),(5)
64. (2),(6),(7)
65. (4),(8),(9)
66. (5),(10),(11)
67. (6),(12),(13)
68. (7),(14),(15)
69. (1),(2)
70. (1),(4),(5)
71. (2),(6),(7)
72. (2),(3)

3 Future Work

We would like to see if the program that was based off the algorithm works for binary trees higher than level 10. We would also want to investigate creating an algorithm for the strict sorting of general trees. As shown in 2.3, there are two different chip firing processes for each tree of varying levels because of how we paired chips for firing in 2.1. This begs the question, can we construct different algorithms that allows for confluence? Another thing to consider is constructing tree graphs that are efficient in terms of how quickly the chip firing process takes so that the chips will sort.

References

[1] Per Bak, Chao Tang, and Kurt Wiesenfeld. Self-organized criticality: An explanation of the 1/ f noise. Phys. Rev. Lett., 59:381–384, 1987

[2] Deepak Dhar. The abelian sandpile and related models. Physica A: Statistical Mechanics and its Applications, 263(1):4 – 25, 1999

[3] Sam Hopkins, Thomas McConville, and James Propp. *Sorting via Chip-Firing*. 2016.