ETE3 and the Collatz Conjecture

...

Pedro Urbina, CEH
@SecureSet

Problem Statement

Start with a number

If the number is one, stop

If the number is even, divide it by two

If the number is odd, multiply by three and add one

Odd numbers become even numbers and alter their factorizations, even numbers lose a factor of two at each iteration until they become odd.

$$f(n) = \begin{cases} n/2 & \text{if } n \equiv 0 \pmod 2 \\ 3n+1 & \text{if } n \equiv 1 \pmod 2 \end{cases}$$

The Big Question...

Do all numbers return to one?

Lets try a few examples...

4 -> 2 -> 1

3 -> 10 -> 5 -> 16 -> 8 -> 4 -> 2 -> 1

15 -> 46 -> 23 -> 70 -> 35 -> 106 -> 53 -> 160 -> 80 -> 40 -> 20 -> 10(!) -> 5 -> 16 -> 8 -> 4 -> 2 -> 1

27 -> 82 -> 41 -> 124 -> 62 -> 31 -> 94 -> 47 -> 142 -> 71 -> 214 -> 107 -> 322 -> 161 -> 484 -> 242 -> 121 -> 364 -> 182 -> 91 -> 274 -> 137 -> 412 -> 206 -> 103 -> 310 -> 155 -> 466 -> 233 -> 700 -> 350 -> 175 -> 526 -> 263 -> 790 -> 395 -> 1186 -> 593 -> 1780 -> 890 -> 445 -> 1336 -> 668 -> 334 -> 167 -> 502 -> 251 -> 754 -> 377 -> 1132 -> 566 -> 283 -> 850 -> 425 -> 1276 -> 638 -> 319 -> 958 -> 479 -> 1438 -> 719 -> 2158 -> 1079 -> 3238 -> 1619 -> 4858 -> 2429 -> 7288 -> 3644 -> 1822 -> 911 -> 2734 -> 1367 -> 4102 -> 2051 -> 6154 -> 3077 -> 9232 -> 4616 -> 2308 -> 1154 -> 577 -> 1732 ->866 -> 433 -> 1300 -> 650 -> 325 -> 976 -> 488 -> 244 -> 122 -> 61 -> 184 -> 92 -> 46 -> 23 -> 70 -> 35 -> 106 -> 53 -> 160 -> 80 -> 40 -> 20 -> 10 -> 5 -> 16 -> 8 -> 4 -> 2 -> 1

Stopping Time and Interesting Sequences

Stopping time is the number of steps for an input to return to one. Twenty seven has a stopping time of 111

Sequence of Stopping Times

0, 1, 7, 2, 5, 8, 16, 3, 19, 6, 14, 9, 9, 17, 17, 4, 12, 20, 20, 7, 7, 15, 15, 10, 23, 10, 111, 18, 18, 18, 106, 5, 26, 13, 13, 21, 21, 21, 34, 8, 109, 8, 29, 16, 16, 16, 104, 11, 24, 24... (A006577)

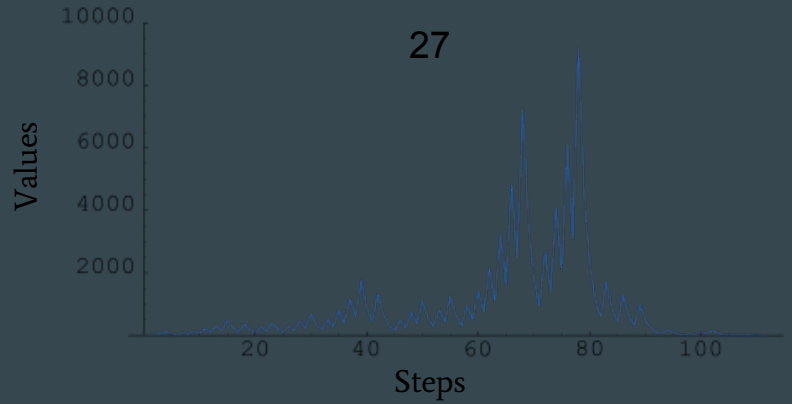
Sequence of Inputs with Largest Stopping Times

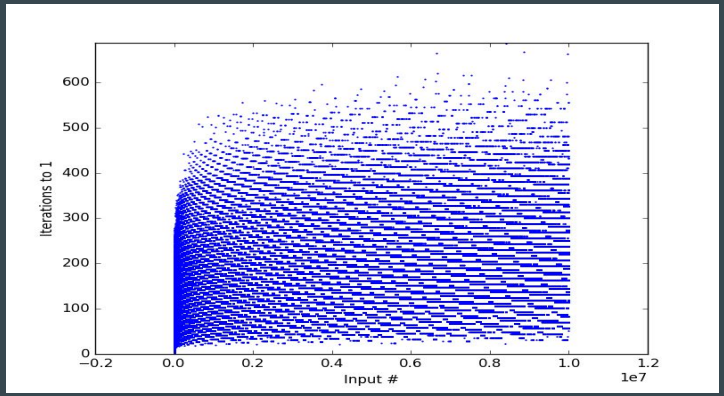
1, 2, 3, 6, 7, 9, 18, 25, 27, 54, 73, 97, 129, 171, 231, 313, 327, 649, 703, 871, 1161, 2223, 2463, 2919, 3711, 6171... (A006877)

Sequence of Inputs with Largest Peak Values

1, 2, 3, 7, 15, 27, 255, 447, 639, 703, 1819, 4255, 4591, 9663, 20895, 26623, 31911, 60975, 77671, 113383, 138367, 159487, 270271, 665215, 704511... (A006884)

Powers of two have a stop time equal to their exponent





Convergent Return Paths and Delay Classes

There exist consecutive inputs which have the same stopping time, how do they converge?

12 -> 6 -> 3 -> 10 -> 5 -> 16 -> 8 -> 4 -> 2 -> 1

13 -> 40 -> 20 -> 10 -> 5 -> 16 -> 8 -> 4 -> 2 -> 1

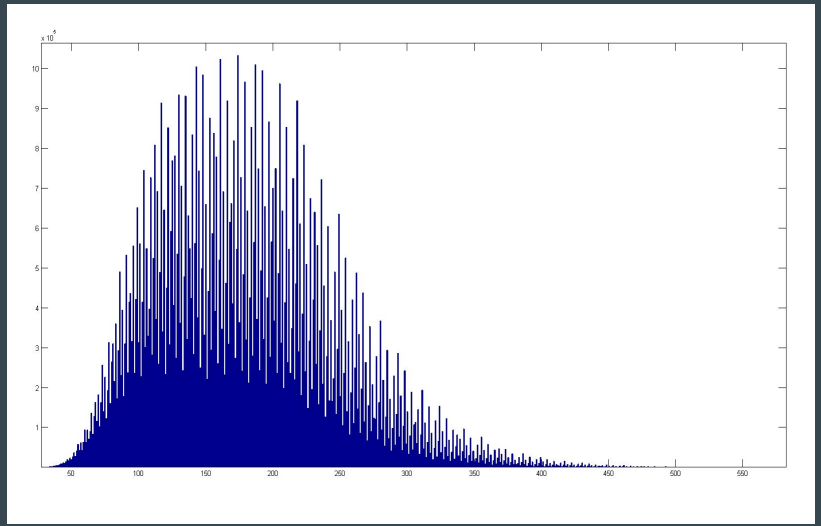
Inputs with the same stopping time are said to be in the same Delay Class, the lowest such number is known as the Class Record.

12 and 13 are both part of Delay Class 9, but 12 is also the Class Record.

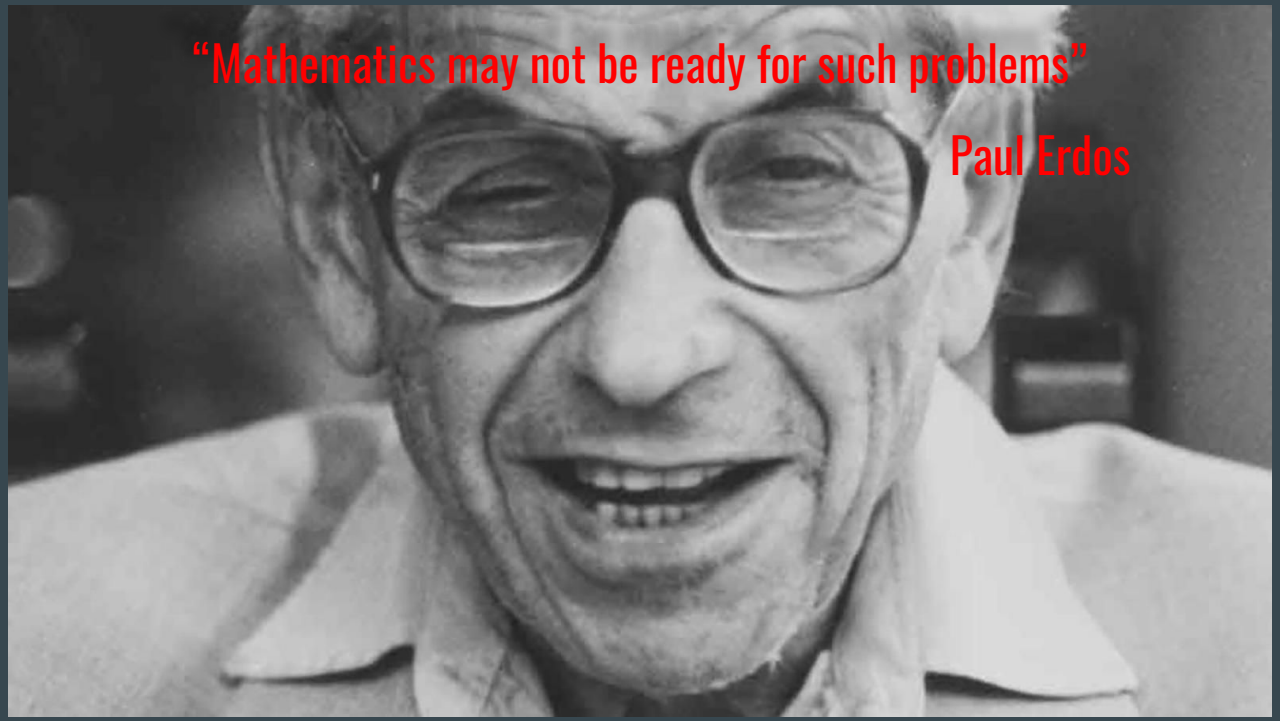
All members of a Delay Class eventually converge, and typically early on.

The largest class record thus far is for class 2258, that number is 279,731,455,495,736,617.

First 100,000 numbers grouped into Delay Classes



But Why



The Naive Implementation

Considerations

- Python
  - Big number support
  - Library availability
  - Generally efficient
- Statistic Logging
  - Steps, Peaks, Orbits
  - Consecutive Runs
  - CSV Output
- Inefficiencies
  - Global Interpreter Lock
  - Duplicative
  - About 30 minutes to compute up to 10M

```
def collatz(k):
 peakValue = 1
 returnSteps = 0
 orbitCount = 0
 growthFactor = 0
 origin = k
 loflag = 0
 hiflag = 0
 while k!= 1:
  if origin < k:
   hiflag = 1
   loflag = 0
  if origin > k:
   loflag = 1
   hiflag = 0
  if k > peakValue:
   peakValue = k
  if k%2 == 0:
   k /= 2
   returnSteps+=1
   if hiflag and (k < origin):
    orbitCount+=1
  else:
   k = k*3 + 1
   returnSteps+=1
   if loflag and (k > origin):
    orbitCount+=1
 growthFactor = peakValue/origin
 data = (origin, returnSteps, peak
 return data
```

```
maximum return steps is 685 and occurs at 8400511
maximum peak value is 60342610919632.0 and occurs at 6631675
maximum orbit count is 113 and occurs at 8546945
maximum growth factor is 9099150.80573641 and occurs at 6631675
maximum step run is 65 at input 5772712
maximum peak run is 16 at input 8965036
maximum orbit run is 2 at input 1
[terra @ versions] $
```

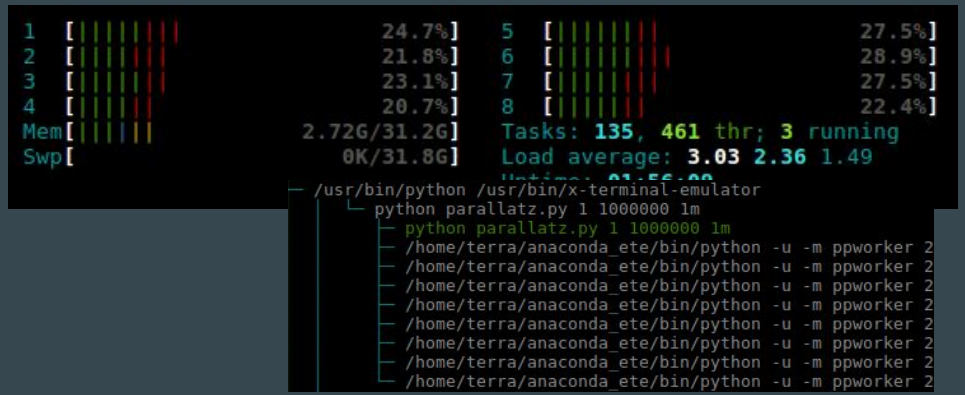
```
  1  [|                                    2.6%]     5  [|||                                  4.7%]
  2  [|||||||||||||||||||||||||||||||||||99.3%]     6  [||                                   4.0%]
  3  [|                                    1.4%]     7  [||                                   2.0%]
  4  [|                                    0.7%]     8  [|                                    1.3%]
  Mem[||||||                          2.35G/31.2G]   Tasks: 121, 486 thr; 3 running
  Swp[                                  0K/31.8G]   Load average: 2.04 2.03 1.44
                                                    Uptime: 00:29:42

  PID USER      PRI  NI  VIRT   RES   SHR S CPU% MEM%   TIME+  Command
 4114 terra      20   0  680M 56624 34024 S  5.4  0.2  0:52.15 ├─ /usr/bin/pytho
 4281 terra      20   0  293M  273M  5668 R 100.  0.9 10:33.70 │  │  └─ python3
 4123 terra      20   0  680M 56624 34024 S  0.0  0.2  0:01.28 │  ├─ /usr/bin/py
 4122 terra      20   0  680M 56624 34024 S  0.0  0.2  0:00.00 │  └─ /usr/bin/py
 3172 terra      20   0  235M 25584 11788 S  0.0  0.1  0:00.13 ├─ /usr/bin/py
 3182 terra      20   0  235M 25584 11788 S  0.0  0.1  0:00.00 │  ├─ /usr/bin
 3181 terra      20   0  235M 25584 11788 S  0.0  0.1  0:00.00 │  └─ /usr/bin
```

```
(18082, 185, 48904.0, 26, 0.010231169118460347, 2.7045680787523505, 0.001437894038270103)
(18081, 92, 54244, 13, 0.005088214147447597, 3.000055306675516, 0.0007189867817045517)
(18082, 185, 48904.0, 26, 0.010231169118460347, 2.7045680787523505, 0.001437894038270103)
(18083, 185, 81376.0, 23, 0.010230603329093623, 4.500138251396339, 0.0012719128463197478)
(18084, 185, 77272.0, 24, 0.010230037602300375, 4.2729484627294845, 0.0013271400132714001)
(18085, 185, 77272.0, 25, 0.010229471938070224, 4.272712192424661, 0.0013823610727121925)
(18086, 185, 77272.0, 28, 0.01022890633639279, 4.272475948247263, 0.0015481587968594493)
(18087, 92, 154528.0, 1, 0.005086526234311937, 8.543594847127771, 5.52883286338254e-05)
(18088, 22, 18088, 0, 0.0012162759840778417, 1.0, 0.0)
(18089, 185, 695464.0, 13, 0.010227209906573055, 38.44679086737796, 0.0007186688042456742)
(18090, 22, 27136.0, 2, 0.0012161415146489773, 1.5000552791597568, 0.0001105583195135434)
```

The Parallel Naive Implementation

Considerations

- pp library for threaded jobs
- tqdm module for loading bar
- CPU Utilization +~100%
- Writing 60GB+ files can crash things
- Still recomputing values :(




```
job_server = pp.Server()

def statlog():
 data_list = []
 lim1 = batchStart
 lim2 = batchStart + 8
 for i in tqdm(range(batchStart,batchEnd)):
  job_list = [job_server.submit(collatz, (i,)) for
  data_list.extend([job() for job in job_list])
  lim1+=8
  lim2+=8
 open(fileName, "w").write("\n".join(data_list))
 job_server.print_stats()
statlog()
```

```
[terra @ versions] $ python parallatz.py 1 10000 10k
100%|| 10000/10000 [00:12<00:00, 786.01it/s]
Job execution statistics:
 job count | % of all jobs | job time sum | time per job | job server
     80000 |        100.00 |      17.4876 |     0.000219 | local
Time elapsed since server creation 12.7268002033
0 active tasks, 8 cores
```

```
[terra @ versions] $ python parallatz.py 1 100000 100k
100%|| 100000/100000 [02:07<00:00, 785.10it/s]
Job execution statistics:
 job count | % of all jobs | job time sum | time per job | job server
    800000 |        100.00 |     179.1408 |     0.000224 | local
Time elapsed since server creation 127.406756878
0 active tasks, 8 cores
```

```
[terra @ versions] $ python parallatz.py 1 1000000 1m
100%|| 1000000/1000000 [21:51<00:00, 762.60it/s]
Job execution statistics:
 job count | % of all jobs | job time sum | time per job | job server
   8000000 |        100.00 |     1865.0076 |     0.000233 | local
Time elapsed since server creation 1311.58688903
0 active tasks, 8 cores
```

Refactoring for Dynamic Growth

The Inverse Problem Statement

In order to explore the Collatz problem with a tree structure it makes sense to grow from existing data, rather than to trace paths to a root.

$$R(n) = \begin{cases} \{2n\} & \text{if } n \equiv 0, 1, 2, 3, 5 \\ \{2n, (n-1)/3\} & \text{if } n \equiv 4 \end{cases} \pmod 6.$$

The inverse function doesn't stop on a condition, so depth of the tree will be the practical limit.

There are also multiple outputs on the alternative condition, so a decision must be made about the order of processing, I decided to use a breadth-first approach with a queue.

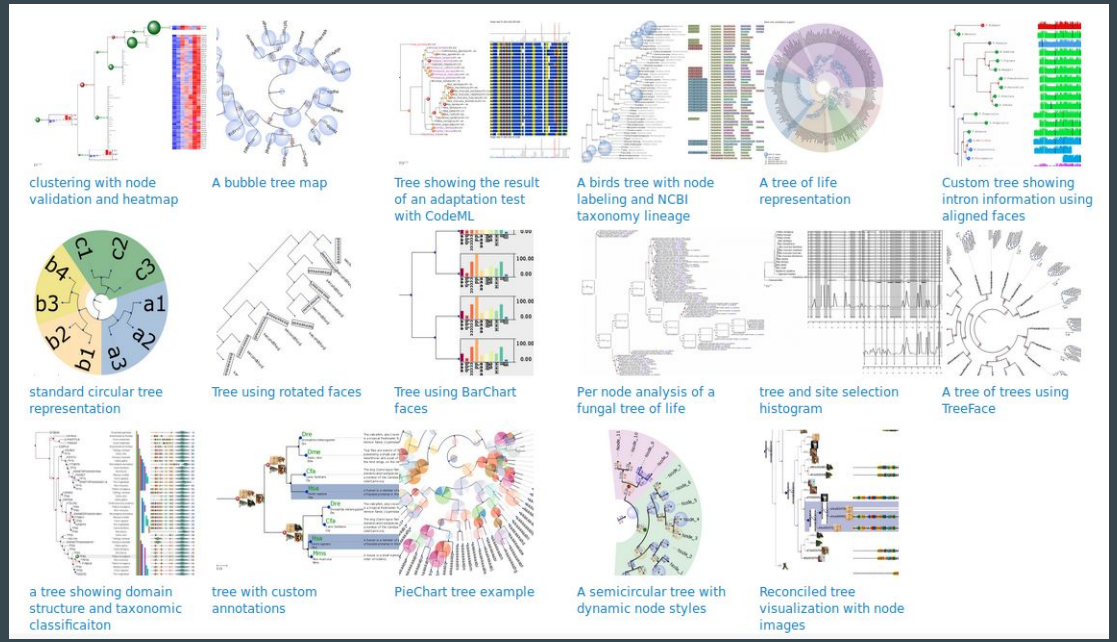
```
depth = int(sys.argv[1])

class Node:
    def __init__(self, value):
        self.value = int(value)
    def norm(node): #next normal collatz node
        self.norm = node
    def inv1(node): #first inverse collatz node
        self.inv1 = node
    def inv2(node): #second inverse collatz node
        self.inv2 = node
    def dist(value): #distance from root
        self.dist = value

class colTree:
    def __init__(self, maxDepth):
        self.root = Node(1)
        self.root.dist = 0
        self.nodeQueue = deque([self.root])
        self.visited = [1]
        self.newickIndex = 2
        self.commaFlag = 0
        self.currentDist = 0

    def grow(self):
        while (nodeQueue):
            self.rCol(nodeQueue.popleft())
```

ETE3 - The (Python) Environment for Tree Exploration

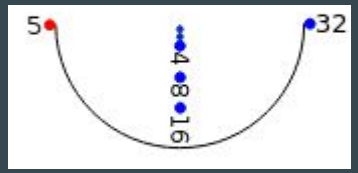
Features

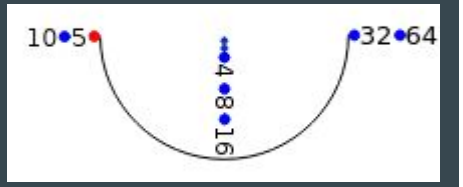
- Dynamic, Programmable, Powerful
- Newick Format
- Rendering Capabilities
- Phylogenetic-specific functions

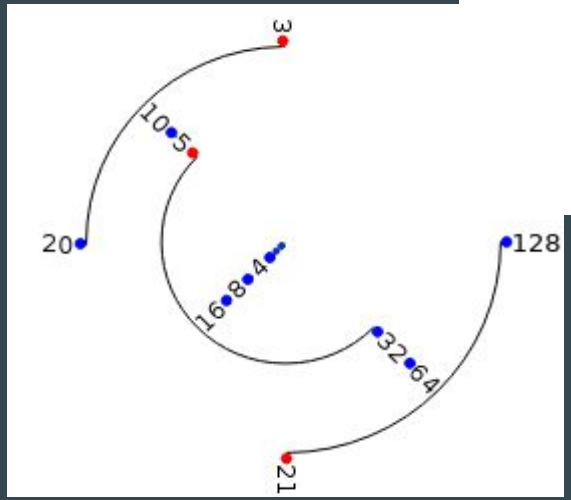
```
from ete3 import Tree
tree = Tree('((A,B), D);')

print tree
#       /-A
#    /-|
# --|   \-B
#    \-D

A = tree & "A"
A.up.show()
```

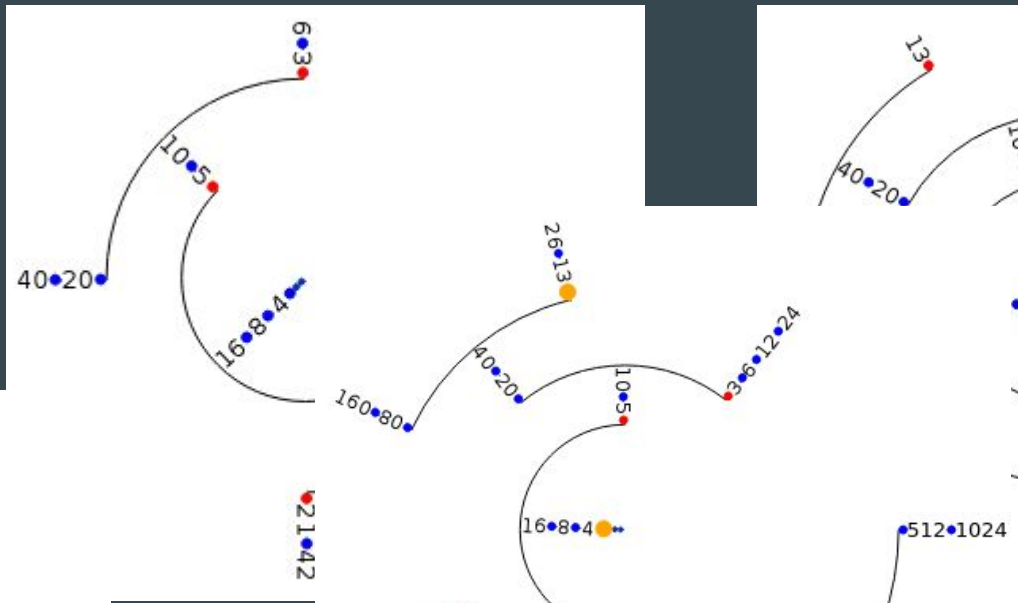


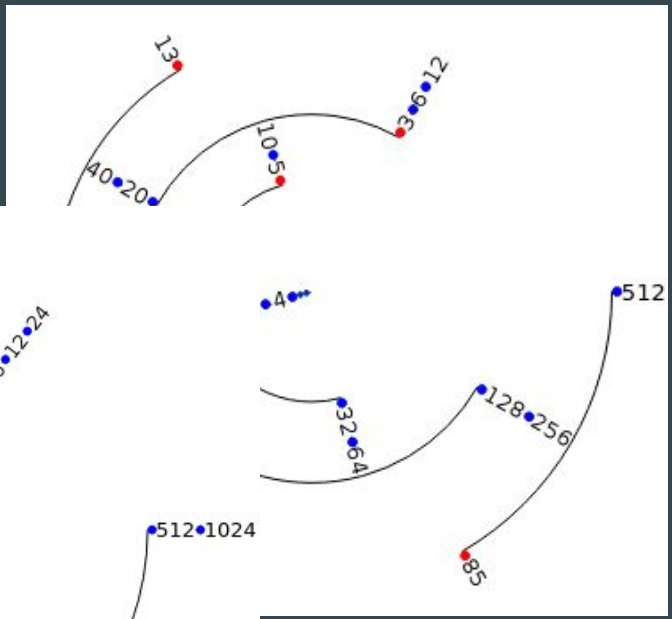
Graphing Time



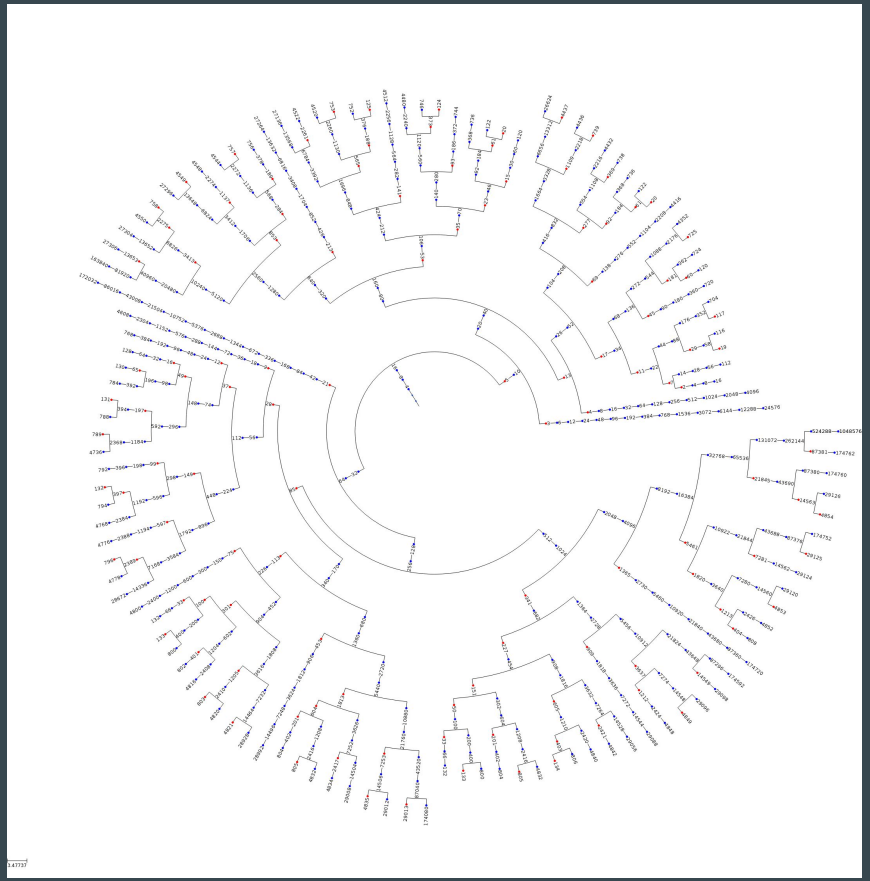


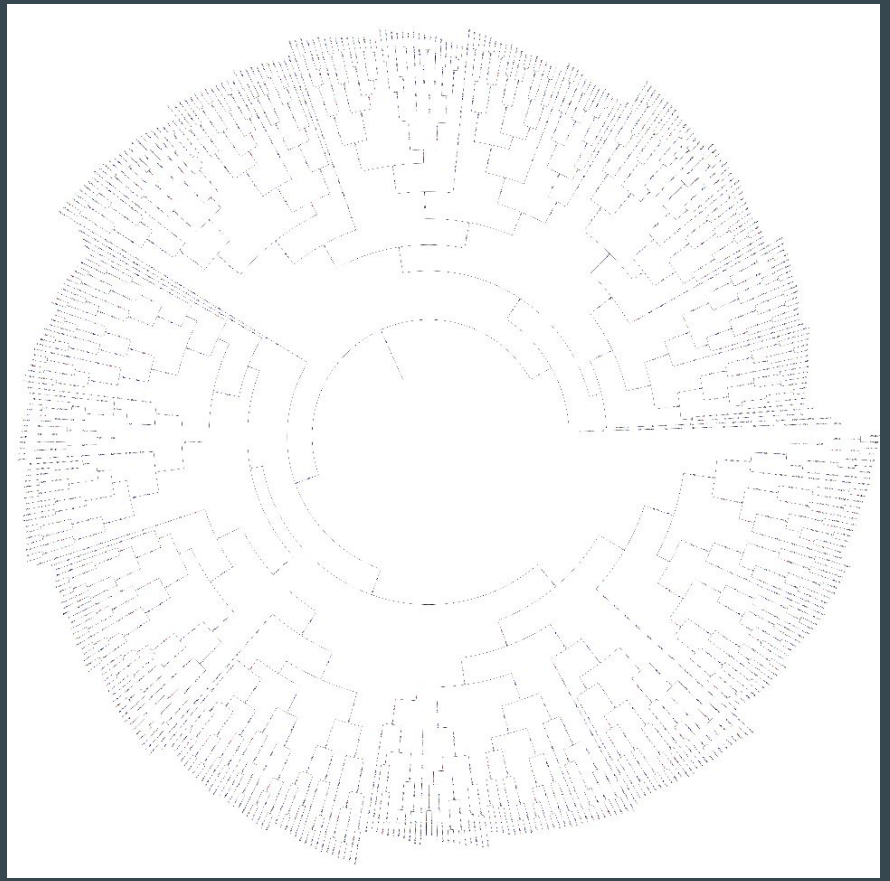


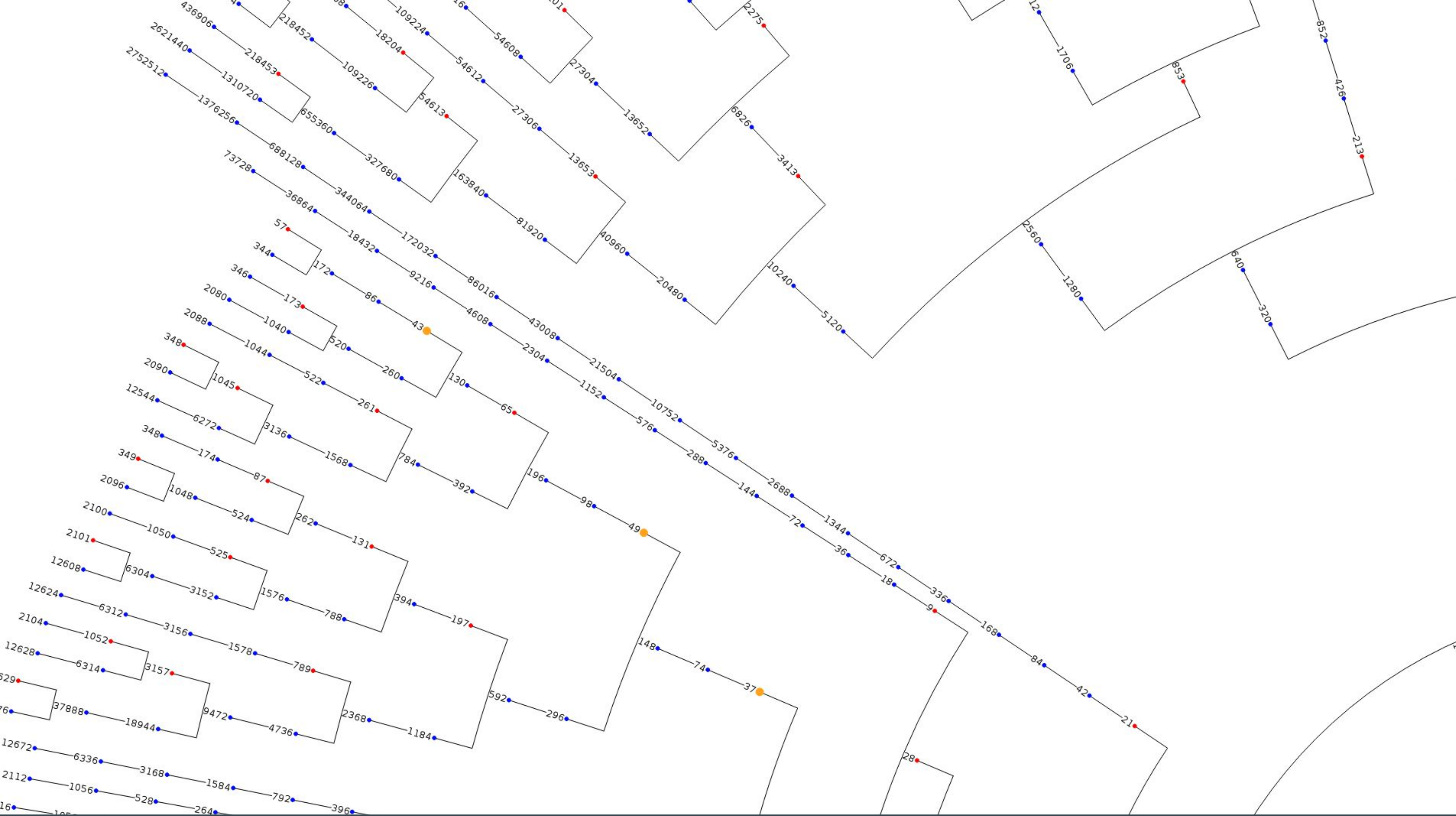




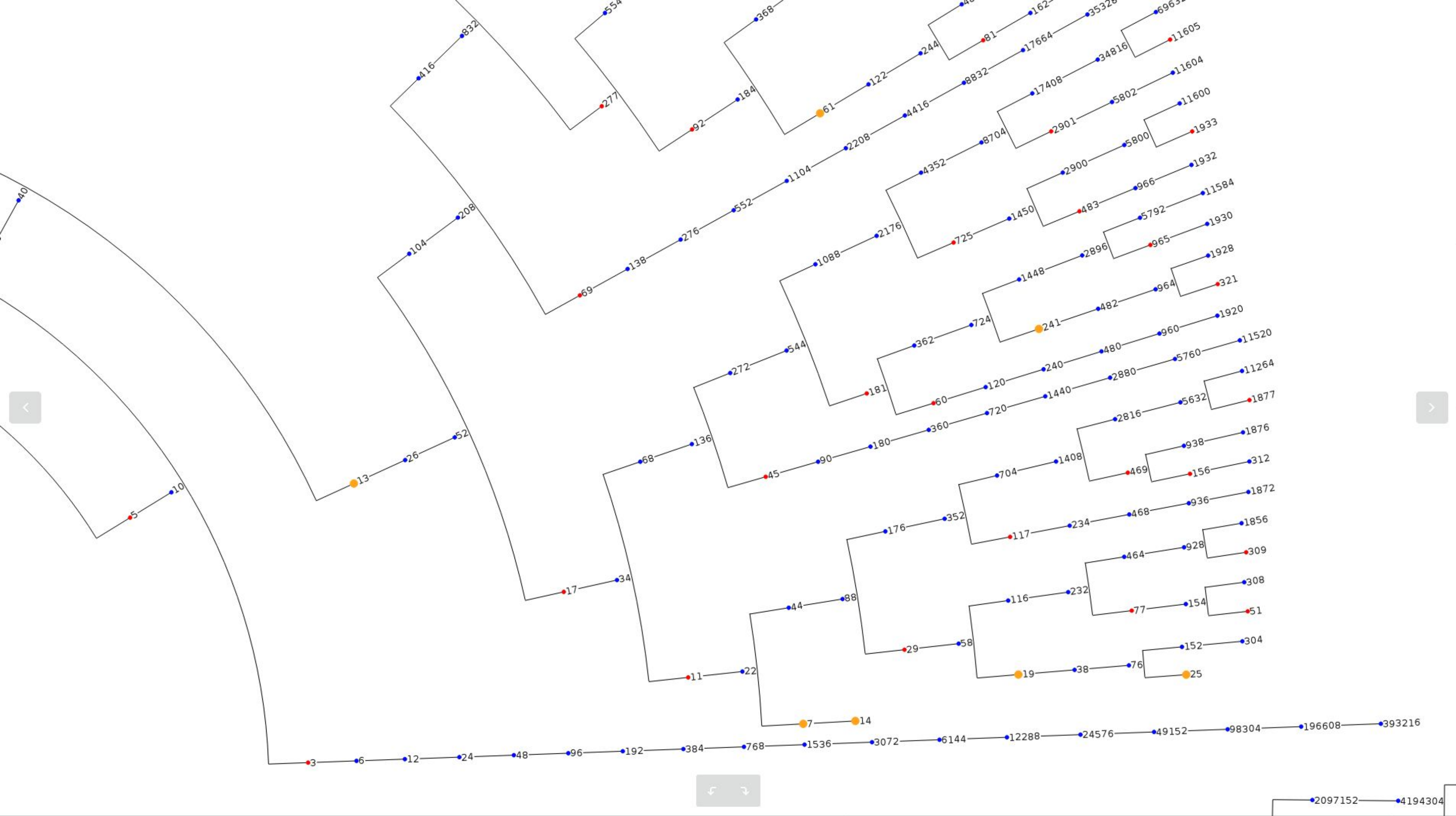
I Like Big Graphs

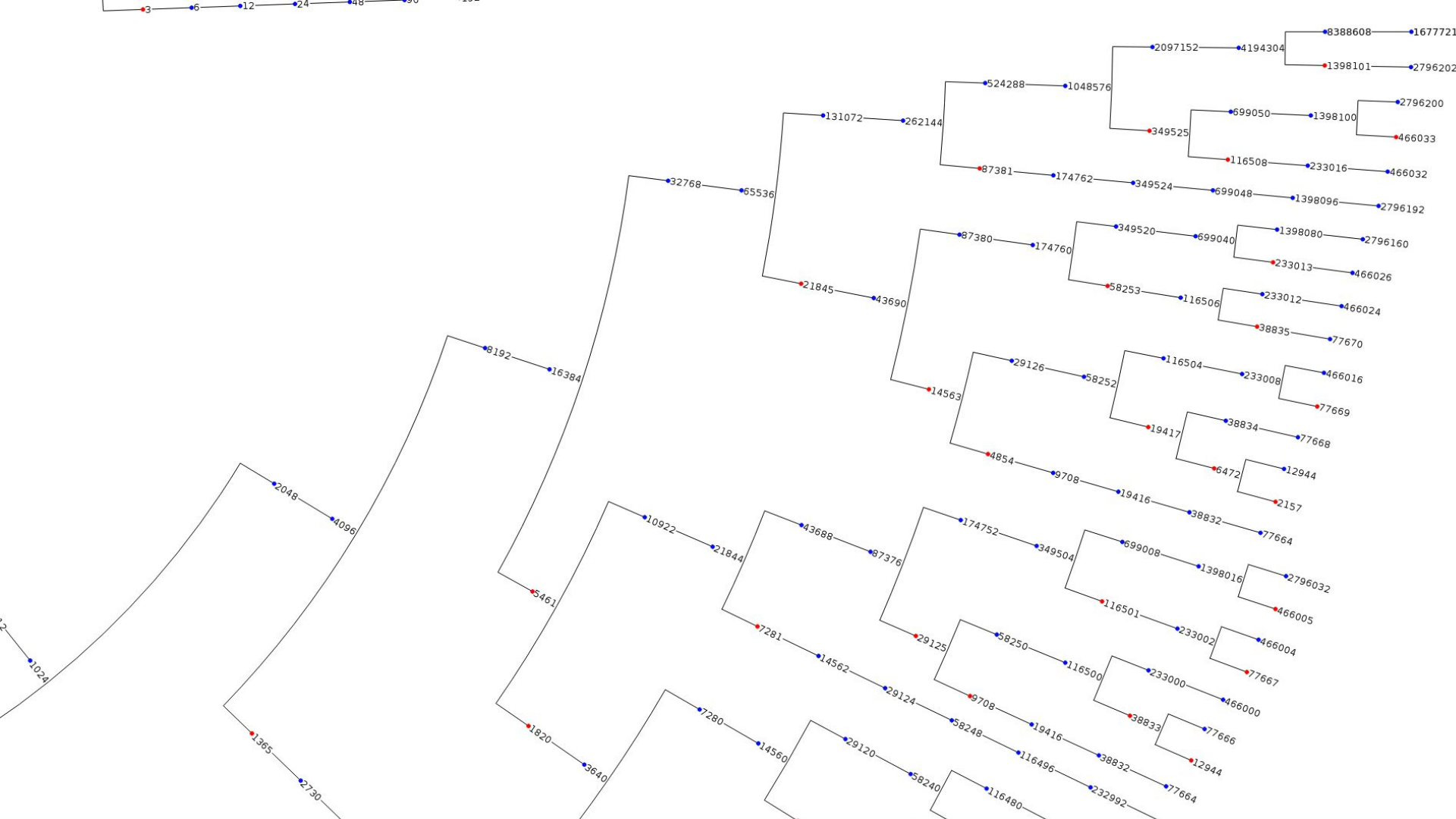


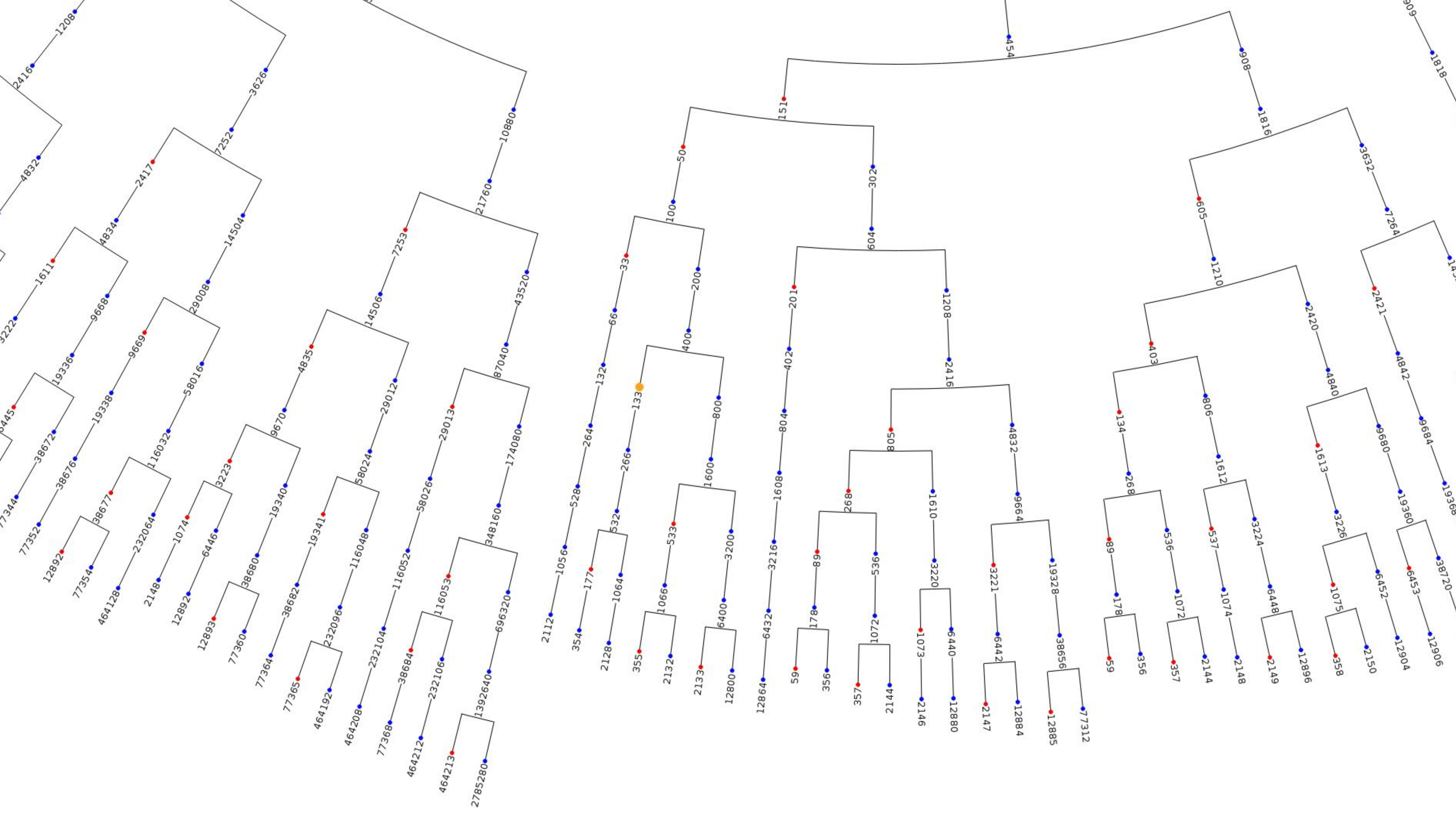




436906
218452
218453
109224
109226
18204
54608
54612
54613
27304
27306
27305
13652
13653
2275
6826
3413
2621440
2752512
1310720
1376256
655360
327680
688128
344064
163840
81920
40960
20480
10240
5120
73728
36864
18432
172032
86016
9216
43008
4608
21504
2304
10752
1152
5376
576
2688
288
1344
144
672
72
336
36
168
18
84
9
42
21
28
1706
853
852
426
213
2560
1280
640
320
57
344
172
86
43
346
173
2080
1040
520
260
130
65
2088
1044
522
261
348
2090
1045
12544
6272
3136
1568
784
392
196
98
49
348
174
87
349
2096
1048
524
262
131
2100
1050
525
2101
12608
6304
3152
1576
788
394
197
12624
6312
3156
1578
789
2104
1052
12628
6314
3157
9472
4736
2368
1184
592
296
148
74
37
37888
18944
12672
6336
3168
1584
792
396
2112
1056
528
264


554
832
368
162
35328
69632
81
11605
416
244
17664
34816
277
184
122
8832
11604
92
61
4416
17408
5802
11600
2208
8704
2901
1933
5800
1104
4352
2900
1932
966
552
1450
483
11584
5792
40
208
276
2176
725
1930
965
104
138
1088
2896
1928
69
1448
321
964
724
482
1920
544
362
241
960
11520
272
480
240
5760
11264
181
120
2880
60
1877
1440
720
5632
136
360
2816
1876
938
68
180
1408
469
312
90
704
156
45
1872
10
13
26
52
352
936
176
468
5
117
234
1856
928
309
464
232
308
17
34
88
116
77
154
51
44
58
29
304
152
76
11
22
19
38
25
7
14
3
6
12
24
48
96
192
384
768
1536
3072
6144
12288
24576
49152
98304
196608
393216
2097152
4194304


3
6
12
24
48
1024
2048
4096
8192
16384
32768
65536
131072
262144
524288
1048576
2097152
4194304
8388608
16777216
1398101
2796202
2796200
466033
699050
1398100
349525
116508
233016
466032
87381
174762
349524
699048
1398096
2796192
21845
43690
87380
174760
349520
699040
1398080
2796160
233013
466026
58253
116506
233012
466024
38835
77670
14563
29126
58252
116504
233008
466016
77669
19417
38834
77668
4854
9708
19416
38832
77664
6472
12944
2157
5461
10922
21844
43688
87376
174752
349504
699008
1398016
2796032
466005
116501
233002
466004
77667
29125
58250
116500
233000
466000
77666
38833
12944
9708
19416
38832
77664
7281
14562
29124
58248
116496
232992
1820
3640
7280
14560
29120
58240
116480
1365
2730



454
151
908
50
302
1816
100
604
605
3632
33
200
201
1208
1210
7264
66
400
402
2416
403
2420
2421
132
133
800
804
805
4832
134
806
4840
4842
264
266
1600
1608
268
1610
9664
268
1612
1613
9680
9684
528
532
533
3200
3216
89
536
3220
3221
19328
89
536
537
3224
3226
19360
19368
1056
177
1064
1066
6400
6432
178
1072
1073
6440
6442
38656
178
1072
1074
6448
1075
6452
6453
38720
2112
354
2128
355
2132
2133
12800
12864
59
356
357
2144
2146
12880
2147
12884
12885
77312
59
356
357
2144
2148
2149
12896
358
2150
12904
12906
1208
2416
3626
7252
4832
2417
4834
14504
10880
21760
7253
43520
14506
1611
9668
29008
3222
19336
9669
58016
4835
29012
87040
6445
38672
19338
116032
9670
29013
174080
58024
77344
38676
38677
232064
3223
19340
58026
348160
77352
12892
77354
464128
1074
6446
2148
12892
38680
19341
116048
116052
116053
696320
12893
77360
38682
232096
232104
38684
232106
1392640
77364
77365
464192
464208
77368
464212
464213
2785280

Thank You

@peddrrrooooo

pedrourbina.com