Project Eueler 107

Kevin Black

May 7, 2018

1 Problem Statement

A k-input binary truth table is a map from k input bits (binary digits, 0 [false] or 1 [true]) to 1 output bit.

How many 6-input binary truth tables, τ, satisfy the formula

$$\tau(a, b, c, d, e, f) \text{ AND } \tau(b, c, d, e, f, a \text{ XOR } (b \text{ AND } c)) = 0$$

for all 6-bit inputs (a, b, c, d, e, f)?

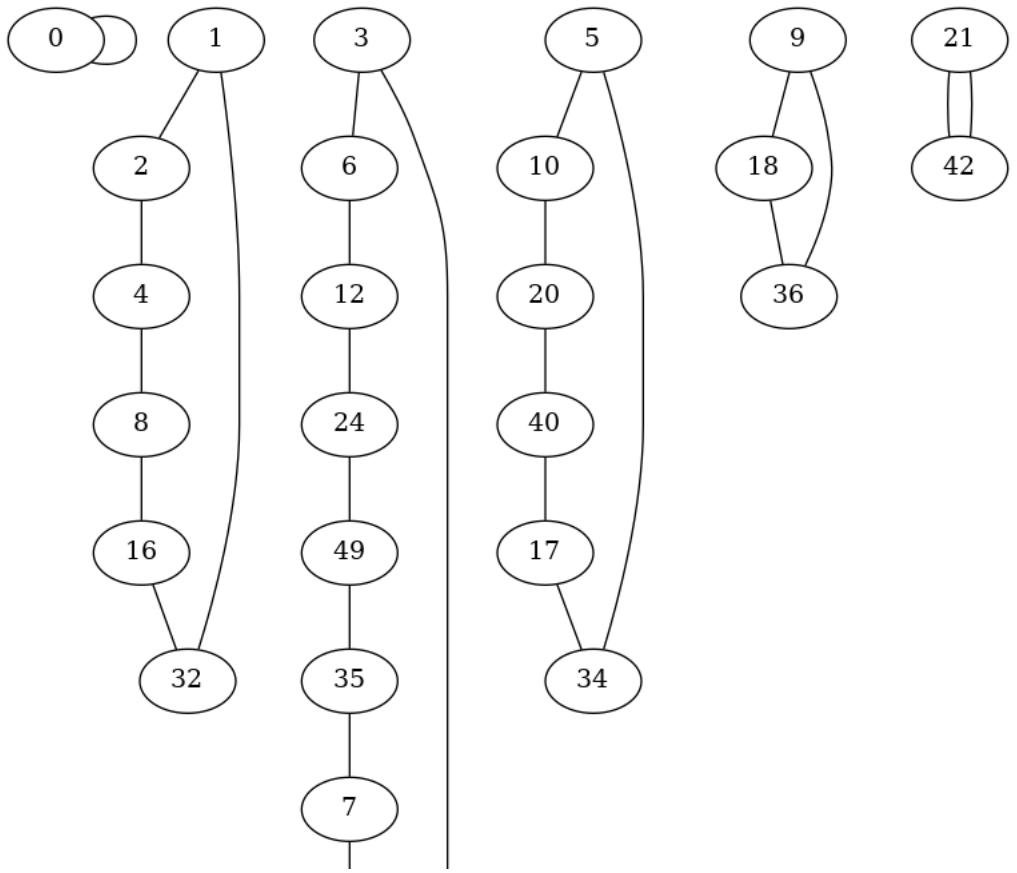
2 Code

```
from itertools import product
from functools import reduce
from operator import mul
from graphviz import Graph

def transform(t):
    # transform (a, b, c, d, e, f) to (b, c, d, e, f, a XOR (b and c))
    return (t[1], t[2], t[3], t[4], t[5], t[0] ^ (t[1] and t[2]))

def ncr(n, r):
    # binomial coefficient
    if n < 0 or r < 0:
        return 0
    r = min(r, n - r)
    numer = reduce(mul, range(n, n - r, -1), 1)
    denom = reduce(mul, range(1, r + 1), 1)
    return numer // denom

def combs(n):
    # calculate number of possibilites for an n-cycle
    return sum(ncr(n - i + 1, i) - ncr(n - i - 1, i - 2) for i in range(n))

inputs = list(product([False, True], repeat=6))

# Generate and render graph of input transformations to observe that they form
    disjoint cycles
dot = Graph()
for i in range(64):
    dot.node(str(i))
dot.edges((str(inputs.index(p)), str(inputs.index(transform(p)))) for p in inputs)
dot.render('graph')

# Count cycle lengths
cycles = []
while inputs:
```

```
    t = inputs.pop()
    n = transform(t)
    length = 1
    while n != t:
        inputs.remove(n)
        n = transform(n)
        length += 1
    cycles.append(length)

# Calculate answer by multiplying together number of possiblities for each cycle
print(reduce(mul, (combs(n) for n in cycles), 1))
```

3 Rationale

First, I decided to examine the transformation $(a, b, c, d, e, f) \rightarrow (b, c, d, e, f, a \text{ XOR } (b \text{ AND } c))$. The line `inputs = list(product([False, True], repeat=6))` creates a list of all possible combinations of 6 binary values (all 64 inputs to the truth table). I then used the graphviz library to examine the nature of this transformation (defined by the method `transform`) on these 64 inputs. The result was the following:



By observing this visualization, it is clear that the input transformation forms 6 disjoint cycle graphs of lengths 1, 2, 3, 6, 6, and 46 (the one of length 46 is cut off in the picture; it continues downward quite a ways).

Each edge in the graph represents a connection between an input (a, b, c, d, e, f) and $(b, c, d, e, f, a$ XOR (b AND c)). In order for (a, b, c, d, e, f) AND $(b, c, d, e, f, a$ XOR (b AND c)) to be 0 for all inputs, no two adjacent nodes in the graph can be 1. The number of ways for this to happen is just the number of ways for this to happen for each cycle multiplied together.

I derived the number of ways to assign 0s and 1s to an n-cycle such that no two adjacent nodes are 1 to be the following:

$$\sum_{k=0}^{n} \binom{n-k+1}{k} - \binom{n-k-1}{k-2}$$

I will explain the derivation on the board. An interesting note is that this expression turns out to just be the nth Lucas number (the Fibonacci sequence with starting values of 2 and 1).

The method `combs` implements this expression naively using the binomial coefficient. The final bit of code finds all the cycles in the transformation, counting their lengths, and then multiplies together the number of possibilites for each cycle to find the answer.