Over the weekend I converted things to python so that in idle, after running a program I can continue to use the variables. I need to make a table of what I tried and what the results were for each case for easier presentation of what worked and what didn’t. Working to see where things slow down. Will run the test with and without a dictionary without saving the values in level 6 to see when things get slow. levelMaker8 will do this test and report the time without making a dictionary and levelMaker9 will do this with a dictionary. levelMaker8 and levelMaker9 does not use any filtering method. They keep every circuit combination they finds. levelMaker10 and levelMaker11 do the same as 8 and 9 except they narrow down the levels by only adding circuits whose truthValues are not the same as level 1 or 2 and who don’t have a subcircuit that is the same as it. With the presence of the dictionary the functions slowed down significantly. Things were slowed down by when we were calculating the truthValues.

Methods of narrowing down the number of items in each level:

1. No narrowing down. Every circuit is kept.
   1. Size of levels 1-5 = [4,6,39,1131,694434]
   2. Going up to level 5 it found 202/256 circuits (0 and 1 included). Attempting to reach level 6 ran into memory problems.
   3. Maximum number of circuits used for these 202 was 9.
2. If we already have something that gives the same truthValues with few number of gates it is thrown out.
   1. This does not ensure the smallest possible circuit for each truthValue, but it is very fast and can find something for each truthValue
   2. The maximum number of gates used in one of the first 202 found by the first 5 levels was 12 as opposed to 9. Maximum number of gates used overall was also 12.
   3. Size of levels 1-8 = [4,6,21,108,141,187,8,0]
   4. There were some cases where it was able to use less gates than Method A for because sometimes things found in higher levels give circuits with less gates. In all cases the circuit found using this method was found in level 6. A couple examples are in 6.18.15\_DailyNotes:
      1. 11100110
         1. Method A used 8 to make ((((a.b).c).(b.0)).(((a.b).(b.c)).(a.b)))
         2. Method B used 7 to make (((((a.b).b).c).((b.c).b)).0)
      2. 10111100
         1. Method A used 8 to make ((((a.b).a).(b.0)).(((a.b).0).(a.c)))
         2. Method B used 7 to make (((((a.c).a).b).((a.b).a)).0)
   5. The presence of items in level 7 does imply that something was found that improved on a previous circuit.
3. Removing anything with more than 12 gates. (in java only)
   1. This was tried because from method B we knew that everything could be solved using at most 12 gates.
   2. This was extremely unsuccessful in terms of reducing the number of items in each level
   3. Size of levels 1-5 = [4,6,39,1131,688044]
4. Anything with truthValues that equal the truthValues of any circuit in either level one or level two is thrown out.
   1. Size of levels 1-5 = [4,6,27,417,94155] (level 5 may be off by a bit but it was 94000<x<94500. The number presented may have been when combined with method C) Combining it with Method C did not help much either.
   2. Significantly reduced the size of level 5 by reducing the sizes of levels 3 and 4, but this was still unable to reach level 6 without running out of memory.
5. Anything with a double negative is thrown out.
   1. In combination with method D this only reduces the size of level 5 by 24 items from what D had. It was not useful.
   2. Size of Method D+E levels 1-5 = [4,6,27,417,94131] (level 5 here may also be off by a bit but it was 94000<x<94500 and only 24 less than method D)
6. If a circuit contains a subcircuit that gives the same truthValues as the circuit itself, throw out the circuit.
   1. This includes anything found in Method E. This should not get rid of anything useful. Suppose circuits p and q have the same truthValues and circuit p is a subcircuit of q that is not q itself. So q must have more gates than p. If you nor circuit x with q you still have to build p to make q and then you can use q to nor with x. Going from p to q will require at least one gate. Instead of adding that one gate doing the nor of p and x will give the same answer with one less gate. If something in q that is in p would simplify the overall circuit, it will still simplify if p is used. If something in q and not in p simplifies with x then then you still need to build p and x, but not the components that simplified in q but not in p. So p is still there.
   2. This was only tested in combination with method D. It reduced the number of circuits in level 5 from what D had, but not previous levels.
   3. Size of Method D+F levels 1-5 = [4,6,27,417,68754]

levelMakers in GateMinimization2.py:

1. Employs method A for finding the levels
   1. Returns the list of levels
   2. Does not complete level 6 can get level 5 in a matter of seconds
2. Employs method D but does so while making a dictionary.
   1. Takes a bit longer to find level 5 then levelMaker1. Cannot find level 6. Size of the lists are smaller (see details for method D).
   2. Returns a dictionary
3. Employs method D while making a dictionary. Stops when the dictionary size reaches 256. This still saves the elements in level 6.
   1. This is the same as levelMakers2 but stops when dictionary size is 256. It never reaches this point. Runs out of memory before it does.
   2. Returns a dictionary
4. This is the same as levelMaker2 except without a dictionary. Employs method D.
   1. Has a small random chance of printing the progression.
   2. Returns a dictionary
5. The same as levelMaker2 .(I didn’t notice I made the same function twice)
   1. a small chance of printing the progress.
   2. Returns dictionary
6. The same as levelMaker2 but does not save the values of level 6. Prints progress with small chance.
   1. It reached 254/256 after >14hrs. (it was stopped before completion)
   2. It does not stop when the dictionary size reaches 256. Does not run into memory problems but is extremely slow.
   3. Returns dictionary
7. Similar to levelMaker2 but uses Method D *and* Method F and makes a dictionary. Saves all levels including level 6+ if it told to run that high.
   1. Will probably run into memory problems.
   2. Ran for 2+ hrs and found about 242/256 of the truthValues
   3. Returns dictionary
8. Uses method A. No narrowing down and *does not* use a dictionary. Does not save level 6 but counts how many operations are performed (or the size level 6 would be) and stops and gives how long it took to reach >=numOperation operations.
   1. Used to test how long it takes to get to a certain point in level 6.
   2. Is very fast. <2 seconds when numOperations is 10^6
   3. ~45 seconds when numOperations = 10^8
   4. Returns a string with the time it took (or the levels if it didn’t go to level 6.)
9. (levelMaker8 with dictionary) Uses method A. No narrowing down and *does* use a dictionary. Does not save level 6 but counts how many operations are performed (or the size level 6 would be) and stops and gives how long it took to reach >=numOperation operations.
   1. Used to test how long it takes to get to a certain point in level 6.
   2. Is slow. 377 seconds when numOperations is 10^5
   3. 527 seconds when numOperations is 10^6
   4. >19min seconds when numOperations = 10^8
   5. Returns a string with the time it took (or the dictionary if it didn’t go to level 6.)
10. (levelMaker 8 with methods D and F) Uses method D+F. No narrowing down and *does not* use a dictionary. Does not save level 6 but counts how many operations are performed (or the size level 6 would be) and stops and gives how long it took to reach >=numOperation operations.
    1. Used to test how long it takes to get to a certain point in level 6.
    2. Is somewhat slow. 109 seconds when numOperations is 10^5
    3. 280 seconds when numOperations is 10^6
    4. Returns a string with the time it took (or the list of levels if it didn’t go to level 6.)
11. (levelMaker 9 with methods D and F) Uses method D+F. No narrowing down and *does* use a dictionary. Does not save level 6 but counts how many operations are performed (or the size level 6 would be) and stops and gives how long it took to reach >=numOperation operations.
    1. Used to test how long it takes to get to a certain point in level 6.
    2. Is somewhat slow. 130 seconds when numOperations is 10^5
    3. 330 seconds when numOperations is 10^6
    4. Returns a string with the time it took (or the dictionary if it didn’t go to level 6.)
12. As it builds circuits, it sorts them into how many gates it uses. It builds circuits from the ones with the fewest number of gates up.
13. Uses method B to find the 256 truth values’ associated circuits
14. **Skipped**

Other things tried:

* Adding the circuits to a file and reading it from the file instead of from java ArrayLists. This ran really slowly(took all night and did not complete levels 6, it made a 335000kb file which word could not open) and a projection of the amount of space needed suggested over 9000gb for level 6

>>> #x = (dict with truthValues mapping to circuits, dict with number of gates mapping to all circuits in them)

>>> len(y)

256

>>> #y is a list of all 256 truth Values possible

>>> #good is a list of all the unfound truthValues that are just one not away from being found

>>> #bad is the list of circuits that when wont be found by simple inversion of what is already had

>>> y = []

>>> for i in range(256):

temp = "{0:b}".format(i)

while(len(temp)<8):

temp = "0"+temp

y.append(temp)

>>> len(y)

256

>>> good = []

>>> bad = []

>>> for key in y:

if key not in x[0]:

print key

if inverter(key) in x[0]:

good.append(key)

print "good"

elif inverter(key) not in x[0]:

bad.append(key)

print "bad"

00010110

bad

00011110

good

00101001

bad

00110110

good

00111110

good

01001001

bad

01010110

good

01011110

good

01100001

bad

01101000

bad

01101001

bad

01101011

bad

01101101

bad

01110110

good

01111001

bad

01111110

good

10000110

bad

10010010

bad

10010100

bad

10010110

bad

10010111

bad

10011110

bad

10110110

bad

10111110

good

11010110

bad

11011110

good

11101001

bad

11110110

good

>>> len(good)

10

>>> len(bad)

18

>>> sortAndPrint2(x[0])

00000000 0

00000001 ((((0.b).(0.a)).0).(0.c))

00000010 ((((c.a).c).0).(c.b))

00000011 ((0.b).(0.a))

00000100 ((((b.a).b).0).(c.b))

00000101 ((0.c).(0.a))

00000110 ((((c.b).c).((c.b).b)).(0.a))

00000111 ((0.a).(c.b))

00001000 (((c.b).0).(b.a))

00001001 ((((0.b).(0.a)).((0.a).c)).((0.b).c))

00001010 ((c.a).c)

00001011 (((c.b).b).(0.a))

00001100 ((b.a).b)

00001101 (((c.b).c).(0.a))

00001110 (((0.c).(0.b)).(0.a))

00001111 a

00010000 ((((b.a).a).0).(c.a))

00010001 ((0.c).(0.b))

00010010 ((((c.a).c).((c.a).a)).(0.b))

00010011 ((0.b).(c.a))

00010100 ((((b.a).b).((b.a).a)).(0.c))

00010101 ((0.c).(b.a))

00010111 ((((c.a).(b.a)).a).(c.b))

00011000 ((((b.a).a).(c.b)).(c.a))

00011001 ((((b.a).b).c).((c.b).b))

00011010 ((((b.a).a).(0.c)).(c.a))

00011011 (((c.b).b).(c.a))

00011100 ((((b.a).b).((b.a).a)).(c.a))

00011101 (((c.b).c).(b.a))

00011111 ((c.a).(b.a))

00100000 (((c.a).0).(b.a))

00100001 ((((0.b).(0.a)).((0.b).c)).((0.a).c))

00100010 ((c.b).c)

00100011 (((c.a).a).(0.b))

00100100 ((((b.a).b).(c.a)).(c.b))

00100101 ((((b.a).a).c).((c.a).a))

00100110 ((((b.a).b).(0.c)).(c.b))

00100111 (((c.a).a).(c.b))

00101000 (((c.b).(c.a)).(b.a))

00101010 ((b.a).c)

00101011 ((((b.a).c).b).(((b.a).c).a))

00101100 (((0.b).(c.a)).(b.a))

00101101 (((((c.b).c).0).(c.a)).(((c.b).c).a))

00101110 (((0.c).(0.b)).(b.a))

00101111 (((c.a).a).(b.a))

00110000 ((b.a).a)

00110001 (((c.a).c).(0.b))

00110010 (((0.c).(0.a)).(0.b))

00110011 b

00110100 ((((b.a).b).((b.a).a)).(c.b))

00110101 (((c.a).c).(b.a))

00110111 ((c.b).(b.a))

00111000 (((0.a).(c.b)).(b.a))

00111001 (((((c.a).c).0).(c.b)).(((c.a).c).b))

00111010 (((0.c).(0.a)).(b.a))

00111011 (((c.b).b).(b.a))

00111100 (((0.b).(0.a)).(b.a))

00111101 (((((c.a).c).0).(c.b)).(b.a))

00111111 ((b.a).0)

01000000 (((b.a).0).(c.a))

01000001 ((((0.b).(0.a)).(b.a)).(0.c))

01000010 ((((c.a).c).(b.a)).(c.b))

01000011 ((((c.a).a).b).((b.a).a))

01000100 ((c.b).b)

01000101 (((b.a).a).(0.c))

01000110 ((((c.a).c).(0.b)).(c.b))

01000111 (((b.a).a).(c.b))

01001000 (((c.b).(b.a)).(c.a))

01001010 (((0.c).(b.a)).(c.a))

01001011 (((((c.b).b).0).(b.a)).(((c.b).b).a))

01001100 ((c.a).b)

01001101 ((((b.a).b).c).((b.a).a))

01001110 (((0.c).(0.b)).(c.a))

01001111 (((b.a).a).(c.a))

01010000 ((c.a).a)

01010001 (((b.a).b).(0.c))

01010010 ((((c.a).c).((c.a).a)).(c.b))

01010011 (((b.a).b).(c.a))

01010100 (((0.b).(0.a)).(0.c))

01010101 c

01010111 ((c.b).(c.a))

01011000 (((0.a).(c.b)).(c.a))

01011001 (((((b.a).b).0).(c.b)).(((b.a).b).c))

01011010 (((0.c).(0.a)).(c.a))

01011011 (((((b.a).b).0).(c.b)).(c.a))

01011100 (((0.b).(0.a)).(c.a))

01011101 (((c.b).c).(c.a))

01011111 ((c.a).0)

01100000 (((c.a).(b.a)).(c.b))

01100010 (((0.c).(b.a)).(c.b))

01100011 (((((c.a).a).0).(b.a)).(((c.a).a).b))

01100100 (((0.b).(c.a)).(c.b))

01100101 (((((b.a).a).0).(c.a)).(((b.a).a).c))

01100110 (((0.c).(0.b)).(c.b))

01100111 (((((b.a).a).0).(c.a)).(c.b))

01101010 ((((b.a).c).(b.a)).(((b.a).c).c))

01101100 ((((c.a).b).(c.a)).(((c.a).b).b))

01101110 ((((b.a).c).(0.b)).(((b.a).c).c))

01101111 (((((b.a).a).0).((b.a).c)).(((b.a).c).c))

01110000 ((c.b).a)

01110001 ((((b.a).a).c).((b.a).b))

01110010 (((0.c).(0.a)).(c.b))

01110011 (((b.a).b).(c.b))

01110100 (((0.b).(0.a)).(c.b))

01110101 (((c.a).c).(c.b))

01110111 ((c.b).0)

01111000 ((((c.b).a).(c.b)).(((c.b).a).a))

01111010 ((((b.a).c).(0.a)).(((b.a).c).c))

01111011 (((((b.a).b).0).((b.a).c)).(((b.a).c).c))

01111100 ((((c.a).b).(0.a)).(((c.a).b).b))

01111101 (((((c.a).c).0).((c.a).b)).(((c.a).b).b))

01111111 ((((b.a).c).c).0)

10000000 (((b.a).c).c)

10000001 ((((0.b).(0.a)).(c.a)).((0.b).c))

10000010 ((((c.a).c).(b.a)).((c.a).b))

10000011 (((c.a).b).((b.a).a))

10000100 ((((b.a).b).(c.a)).((b.a).c))

10000101 (((c.a).a).((b.a).c))

10000111 ((((c.b).0).(b.a)).((c.b).a))

10001000 (c.b)

10001001 ((((0.b).(0.a)).(c.b)).((c.b).c))

10001010 (((b.a).a).c)

10001011 (((c.b).b).((c.b).a))

10001100 (((c.a).a).b)

10001101 (((c.b).c).((c.b).a))

10001110 (((0.c).(0.b)).((c.b).a))

10001111 (((c.b).a).0)

10010000 ((((b.a).a).(c.a)).((b.a).c))

10010001 (((c.a).b).((b.a).c))

10010011 ((((c.a).0).(b.a)).((c.a).b))

10010101 ((((b.a).0).(c.a)).((b.a).c))

10011000 ((((b.a).a).(c.b)).((c.b).c))

10011001 (((c.b).c).((c.b).b))

10011010 (((((b.a).a).c).((b.a).a)).((((b.a).a).c).c))

10011011 ((((c.a).0).(c.b)).((c.b).b))

10011100 (((((c.a).a).b).((c.a).a)).((((c.a).a).b).b))

10011101 ((((b.a).0).(c.b)).((c.b).c))

10011111 ((((c.a).(b.a)).(c.a)).(((c.a).(b.a)).(b.a)))

10100000 (c.a)

10100001 ((((0.b).(0.a)).(c.a)).((c.a).c))

10100010 (((b.a).b).c)

10100011 (((c.a).b).((c.a).a))

10100100 ((((b.a).b).(c.a)).((c.a).c))

10100101 (((c.a).c).((c.a).a))

10100110 (((((b.a).b).c).((b.a).b)).((((b.a).b).c).c))

10100111 ((((c.b).0).(c.a)).((c.a).a))

10101000 (((c.b).(c.a)).c)

10101001 (((((0.b).(0.a)).c).((0.b).(0.a))).((((0.b).(0.a)).c).c))

10101010 (0.c)

10101011 (((0.c).b).((0.c).a))

10101100 (((0.b).(c.a)).((c.a).a))

10101101 (((((c.a).c).0).(c.b)).((c.a).a))

10101110 (((0.c).(0.b)).((0.c).a))

10101111 (((c.a).a).0)

10110000 (((c.a).b).a)

10110001 (((c.a).c).((c.a).b))

10110010 (((0.c).(0.a)).((c.a).b))

10110011 (((c.a).b).0)

10110100 (((((c.b).b).a).((c.b).b)).((((c.b).b).a).a))

10110101 ((((b.a).0).(c.a)).((c.a).c))

10110111 ((((c.b).(b.a)).(c.b)).(((c.b).(b.a)).(b.a)))

10111000 (((0.a).(c.b)).((c.b).b))

10111001 (((((c.a).c).0).(c.b)).((c.b).b))

10111010 (((0.c).(0.a)).((0.c).b))

10111011 (((c.b).b).0)

10111100 (((((c.a).b).a).((c.a).b)).((((c.a).b).a).a))

10111101 (((((c.a).c).((c.a).b)).((c.a).b)).((((c.a).c).((c.a).b)).a))

10111111 ((((b.a).0).(c.a)).0)

11000000 (b.a)

11000001 ((((0.b).(0.a)).(b.a)).((b.a).c))

11000010 ((((c.a).c).(b.a)).((b.a).b))

11000011 (((b.a).b).((b.a).a))

11000100 (((b.a).c).b)

11000101 (((b.a).c).((b.a).a))

11000110 (((((c.a).c).b).((c.a).c)).((((c.a).c).b).b))

11000111 ((((c.b).0).(b.a)).((b.a).a))

11001000 (((c.b).(b.a)).b)

11001001 (((((0.c).(0.a)).b).((0.c).(0.a))).((((0.c).(0.a)).b).b))

11001010 (((0.c).(b.a)).((b.a).a))

11001011 (((((b.a).b).0).(c.b)).((b.a).a))

11001100 (0.b)

11001101 (((0.b).c).((0.b).a))

11001110 (((0.c).(0.b)).((0.b).a))

11001111 (((b.a).a).0)

11010000 (((b.a).c).a)

11010001 (((b.a).c).((b.a).b))

11010010 (((((c.b).c).a).((c.b).c)).((((c.b).c).a).a))

11010011 ((((c.a).0).(b.a)).((b.a).b))

11010100 (((0.b).(0.a)).((b.a).c))

11010101 (((b.a).c).0)

11010111 ((((c.b).(c.a)).(c.b)).(((c.b).(c.a)).(c.a)))

11011000 (((0.a).(c.b)).((c.b).c))

11011001 (((((b.a).b).0).(c.b)).((c.b).c))

11011010 (((((b.a).c).a).((b.a).c)).((((b.a).c).a).a))

11011011 (((((b.a).c).((b.a).b)).((b.a).c)).((((b.a).c).((b.a).b)).a))

11011100 (((0.b).(0.a)).((0.b).c))

11011101 (((c.b).c).0)

11011111 ((((c.a).0).(b.a)).0)

11100000 (((c.a).(b.a)).a)

11100001 (((((0.c).(0.b)).a).((0.c).(0.b))).((((0.c).(0.b)).a).a))

11100010 (((0.c).(b.a)).((b.a).b))

11100011 (((((b.a).a).0).(c.a)).((b.a).b))

11100100 (((0.b).(c.a)).((c.a).c))

11100101 (((((b.a).a).0).(c.a)).((c.a).c))

11100110 (((((b.a).c).b).((b.a).c)).((((b.a).c).b).b))

11100111 (((((b.a).c).((b.a).a)).((b.a).c)).((((b.a).c).((b.a).a)).b))

11101000 (((0.b).(c.a)).((0.a).(c.b)))

11101010 (((0.c).(b.a)).0)

11101011 (((((0.c).b).((0.c).a)).((0.c).b)).((((0.c).b).((0.c).a)).((0.c).a)))

11101100 (((0.b).(c.a)).0)

11101101 (((((0.b).c).((0.b).a)).((0.b).c)).((((0.b).c).((0.b).a)).((0.b).a)))

11101110 (((0.c).(0.b)).0)

11101111 (((((b.a).a).0).(c.a)).0)

11110000 (0.a)

11110001 (((0.a).c).((0.a).b))

11110010 (((0.c).(0.a)).((0.a).b))

11110011 (((b.a).b).0)

11110100 (((0.b).(0.a)).((0.a).c))

11110101 (((c.a).c).0)

11110111 ((((c.b).0).(b.a)).0)

11111000 (((0.a).(c.b)).0)

11111001 (((((0.a).c).((0.a).b)).((0.a).c)).((((0.a).c).((0.a).b)).((0.a).b)))

11111010 (((0.c).(0.a)).0)

11111011 (((((b.a).b).0).(c.b)).0)

11111100 (((0.b).(0.a)).0)

11111101 (((((c.a).c).0).(c.b)).0)

11111110 (((((0.b).(0.a)).0).(0.c)).0)

11111111 1