

Re: Collatz at the Hilbert Hotel

Source: <http://sci.tech-archive.net/Archive/sci.math/2005-01/2300.html>

mensanator_at_aol.compost

Date: 01/08/05

Date: 7 Jan 2005 18:42:10 -0800

Michael Collins wrote:

<snip>

> *As an addendum if the duplicators are programmed so that the creation of the Drones is a function of the neighbouring rooms and have Drones using similar rules about whether to remain in the room or leave (which must be nicer than being 'bopped') then Hilbert's Hotel will instead function as a CA and, from the CAs in Wolfram's ANKOS, the hotel can be made to function as a Turing Machine.*

<snip>

>
> *Any comments on this will be gratefully received.*

As an addendum to my previous post where I pointed out the equivalence of the Hilbert algorithm to the standard Collatz algorithm, I've gone ahead and made a Cellular Automata version of the Collatz algorithm that does indeed, operate on neighboring cells and is completely independent of any mathematical reference point. By contrast, the standard Collatz algorithm always anchors the least significant 1 bit to 2^{*0} while the Hilbert algorithm is constantly shifting the least significant 1 bit anchor to ever higher powers of 2.

Here's the Python program:

```
## Collatz Cellular Automata
##
## One dimensional CA with edge wrapping
##
#
```

```

#
#def print_ca():
# ca_out = ""
# for i in ca:
# ca_out = ca_out + i
# print ca_out
#
#
## ca is the one-dimensional array of cells.
#
#ca = []
#
## An infinite array is a bit tricky, so we'll
## make a finite array and use edge wrapping.
## Size should be chosen to be bigger than the
## bit length of the Excursion.
#
#for i in xrange(50):
# ca.append('0')
#
## Drop the life-form into the array at some
## arbitrary point. In this example, the life-form
## will be 27 (11011 in binary).
#
#ca[18] = '1'
#ca[19] = '1'
#ca[20] = '0'
#ca[21] = '1'
#ca[22] = '1'
#
## The CA rules require that we know the starting point
## of the life-form. Without a mathematical reference,
## we'll get lost otherwise when the life-form sails off
## the left end of the array and wraps around to the right end.
#
#start = 22
#
## Initialize the Full Adder lookup tables
##
## a Full Adder adds two binary operands with carry
#
#fa_sum =
{'000':'0','001':'1','010':'1','011':'0','100':'1','101':'0','110':'0','111':'1'}
#fa_car =
{'000':'0','001':'0','010':'0','011':'1','100':'0','101':'1','110':'1','111':'1'}
#
#
## A cellular automata just runs forever, so we'll kill it after a
## finite number of generations.
#
#done = 62

```

```

#
#while done>0:
# # each line of output is one generation of the automata
# print_ca()
#
# i = start
#
# # by pre-setting the Carry Bit, we don't need a separate
# # incrementing step
# C = '1'
#
# # as we do the Full Adder arithmetic, we need to find the
# # first 1 bit that appears in the Sum. This will be where we
# # start the next generation
# next_start = -1
#
# # in each generation, we'll process every cell in the array
# field = len(ca)
#
# for j in xrange(field-1):
# # operand A is the current bit
# A = ca[i]
#
# # operand B is the bit to the left of operand A.
# # this is, of course operand A multiplied by 2.
# # added together, A and B become "3n" with the
# # pre-set Carry Bit providing the "+1"
# B = ca[i-1]
#
# # take the Carry Bit (C), operand A (A) and operand B (B)
# # and make a key for the Full Adder lookup tables
# FA = C + A + B
#
# # S is the Sum
# S = fa_sum[FA]
#
# # the carry out from the Full Adder becomes the carry-in
# # for the next bit position
# C = fa_car[FA]
#
# # if we get a '1' in the Sum and it's the first,
# # remember where it occurred
# if (S=='1') and (next_start==-1):
# next_start = i
#
# # update the array
# ca[i] = S
#
# i -= 1
#
# # from the 0 index, Python can reach around the edge with

```


