Counting unholey polyominoes

A000104 was taken to 40 terms (from a previous set of 28 terms) by using the following formula:

(4*A056879(n) + 4*A056881(n) + 4*A056883(n) + 6*A056880(n) + 6*A056882(n) + 6*A357647(n) + 7*A357648(n) + A006724(n)) / 8

Each of the first 7 terms of the formula corresponds to a specific symmetry of unholey polyomino. The last term corresponds to the number of fixed self-avoiding polygons on the square lattice, which corresponds to the number of fixed unholey polyominoes. This latter sequence has already been calculated through a(42) by Iwan Jensen.

It was therefore necessary to calculate the symmetries through to at least 40 terms, beyond the 28 terms already available. The last two sequences, A35764x, were not present in OEIS.

Sequence	Symmetry	Subdivision	Program
A056879	Mirror 90	M90C: Axis thru square centre	countm90c
		M90V: Axis thru vertex	countm90v
A056881	Mirror 45	M45	countm45
A056883	Rotational 180	R180C: About centre of square	countr180cf:startersize=1x1
		R180ME: About mid-edge of square	countr180cf:startersize=2x1
		R180V: About vertex	countr180cf:startersize=2x2
A056880	Both mirror 90	2M90C: Axis thru square centre	countE
		2M90ME: Axis thru mid-edge of square	countdomino
		2M90V: Axis thru vertex	count2m90v
A056882	Both mirror 45	2M45C: About centre of square	countm45
		2M45V: About vertex	countm45
A357647	Rotational 90	R90C: About centre of square	countr90ccf
		R90V: About vertex	countr90vcf
A357648	All	AllC: About centre of square	countallccf
		AllV: About vertex	countallvcf

These are:

Program runtimes:

Program	Size	Time
countm90c	40	63 hrs
countm90v	40	14 hrs
countm45	40	21 hrs
countr180cf:startersize=1x1	39	1-2 days
countr180cf:startersize=2x1	40	35 hrs
countr180cf:startersize=2x2	40	13 hrs
countE	52	30 secs
countdomino	54	18 secs
count2m90v	56	2 secs
countr90ccf	57	26 secs
countr90vcf	56	21 secs
countallccf	57	< 1 sec
countallvcf	60	< 1 sec

Runtimes tend to multiply by 4 for each increase in 2 of the size.

Unfortunately, there is no logging as the programs run. It would be reassuring to have a line on stderr with a timestamp for every 10000, 100000 or million polyominoes counted.

The classes used are:

Program	Class	Extends	Description
countm90c	CounterM90C	CounterBase	Outputs M90C
			numbers in OEIS
			format
countm90v	CounterM90V	CounterBase	Outputs M90V
			numbers in OEIS
			format
countm45	CounterM45	CounterWithMultipleCopies	Outputs M45, M45C,
			M45V numbers as csv
countr180cf	CounterR180CentreFull	Counter	Outputs R180
			numbers in OEIS
			format, according to
			input parameter
			startersize
countEnew	CounterLetterECentreFull	CounterBase	Outputs 2M90C
			numbers in OEIS
			format
countdomino	CountDominoCentre2M90	CounterBase	Outputs 2M90ME
			numbers in OEIS
			format
count2m90v	Counter2M90VCentreFull	CounterBase	Outputs 2M90V
			numbers in OEIS
			format
countr90ccf	CounterR90CCentreFull	CounterWithMultipleCopies	Outputs 2M90V
			numbers in OEIS
			format
countr90vcf	CounterR90VCentreFull	CounterWithMultipleCopies	Outputs 2M90V
			numbers in OEIS
			format
countallccf	CounterAllCCentreFull	CounterBase	Outputs 2M90V
			numbers in OEIS
			format
countallvcf	CounterAllVCentreFull	CounterBase	Outputs 2M90V
			numbers in OEIS
			format

All the programs require parameters:

size=n : the target size for the program run

unholey=true : to specify we are looking for unholey polyominoes; precisely what output is obtained with unholey=false or missing is not guaranteed.

The goal would have been for all classes to extend CounterBase but this has not yet been achieved.

Each class that extends CounterBase works on the same principles:

- 1. The class starts up the polyomino with the initial centre square or squares.
- 2. It then calls the recursion method of CounterBase.
- 3. This latter performs the recursion to obtain polyomino growth by:
 - a. Choosing the first available candidate growth square and then generating recursively all of its sons, and then:
 - b. Eliminating said square as a candidate and then generating recursively all polyominoes that do not have it.
- 4. CounterBase may invoke class-specific methods useful to define which candidate squares should be considered, which symmetries should be included, etc.

The other, non-CounterBase classes work on the same principles but with their own ad hoc code.

All of the classes use a double representation of the current polyomino:

- An array root[][] where root[0][i] contains the x coordinate of the i-th square, and root[1][i] the y coordinate
- An array board[][] where each position contains an indication of the presence of a square in the polyomino

As each class is dedicated to the counting of some polyomino symmetry, often only a quarter, half or eighth of the complete polyomino will be represented. This complicates the counting of holes, as can be seen in the code.

The number of holes is obtained through the following formula (based on a generalization of Pick's Theorem): H = n + 1 - i - s / 2, where:

- n is the size (area) of the polyomino;
- i is the number of completely internal vertices; i.e., the number of vertices that are surrounded by 4 squares;
- s is the number of vertices on a single border; i.e., vertices that are the corners of 1, 2 or 3 squares, but excluding those that touch only 2 squares that are diagonally adjacent.

The algorithm for calculating the parameters needed by the formula involves considering each vertex of each square. Each represented square may correspond though to more than 1 effective square in the complete polyomino, and so there will be a multiplier to use for each addition to the "I" and "s" parameters (see getHoleyMultiplier).

Further, it is sometimes necessary to consider the existence of squares that are not represented. Each class must therefore offer a way of checking the occupancy of a virtual square by mapping its coordinates onto those of a represented square (see containsSpecific).

Most of the classes add 1 to an array called counter2 for each unholey polyomino found. During the phase of results presentation, some classes may need to divide the counters by 2 or 4 to take into account that the algorithm may find the same polyominoes multiple times in different rotational or reflectional positions.

Each program should be run according to this example, where the trailing parameter "none" is useless but present for historical reasons.

java -jar UnholeyCounters.jar count2m90v:size=56,unholey=true none

There may be some upper/lowercase discrepancies between the above tables and the actual implementation.

John Mason October 11, 2022

For any further information, please write to me at the email address :

TheIllustratedPolyomino (at) gmail (dot) com