Nurikabe puzzle

Zhen Zuo

ABSTRACT

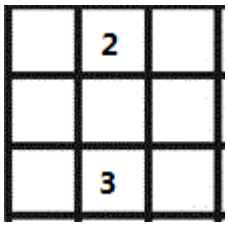
Single-player games (often called puzzles) have received considerable attention from the scientific community. Consequently, interesting insights into some puzzles, and into the approaches for solving them, have emerged. In this article, I focus on Nurikabe puzzle and try to find some pattern of it.

1. INTRODUCTION

Nurikabe (hiragana: ぬりかべ) is a binary determination puzzle named for Nurikabe, an invisible wall in Japanese folklore that blocks roads and delays foot travel. This article mainly talks about rules and strategies that can help readers to solve this kind of puzzle. Then we talk about some computer science programing that can possibly solve puzzles too. In addition, we look into how many possible Nurikabe puzzles exist. Finally, introduce some other kinds of Nurikabe.

2. RULES

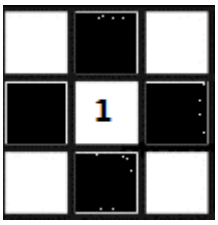
a) Island cells connect left/right and up/down, but not diagonally. The same is true of ocean cells.
b) Each island must contain exactly one numbered cell, which describes its area in number of cells.
c) When the puzzle is done, all the ocean cells must be connected.
d) No 2×2 cells can be completely ocean.
e) No blind guessing should be required to solve Nurikabe puzzle. (This rule is very important. It makes Nurikabe puzzle easier and every step should be done with logic not hypothesis. Many other puzzles do not have this property, such as Sudoku. We only can make simple hypothesis, such as one step to test whether it obey the rule and we should not make a very long hypothesis to determine further cells, which will only make the problem much more complex)
f) The answer should be unique.



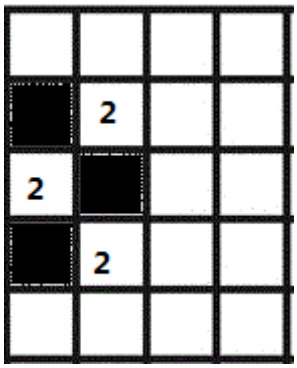
This is not a Nurikabe because the answer is not unique.

3. STRATEGY

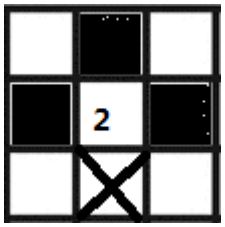
a) 1 must be surrounded by oceans. This is the first step to solve this puzzle.



b) When we find two island lies diagonally, then there must be oceans near the diagonally. This is the second step to solve this puzzle.

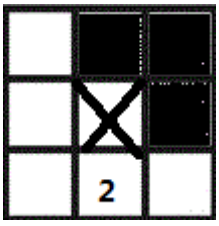


c) When the number in the cell is greater than one and three cells near it are ocean and another one is unknown, then the unknown cell near it must be island.

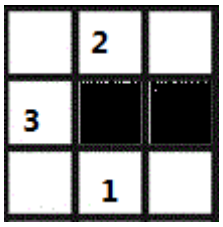


(⊠ is the island.)

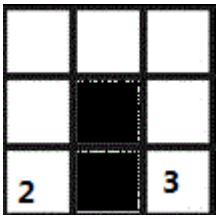
d) When there are three oceans and one unknown cell in 2×2 cells, the unknown cell must be island. (This is especially useful when solving big size puzzle. It means sometimes we need to use island to brake the 2×2 oceans. This usually can control the direction of islands)



e) When an ocean is at the edge of a map and was surrounded by two islands, or



the ocean is surrounded by three islands, then the only other side of the cell must be ocean.



(Note: All the graphs above are only a small part of a large size of puzzle, they are only created to help to explain those strategies)

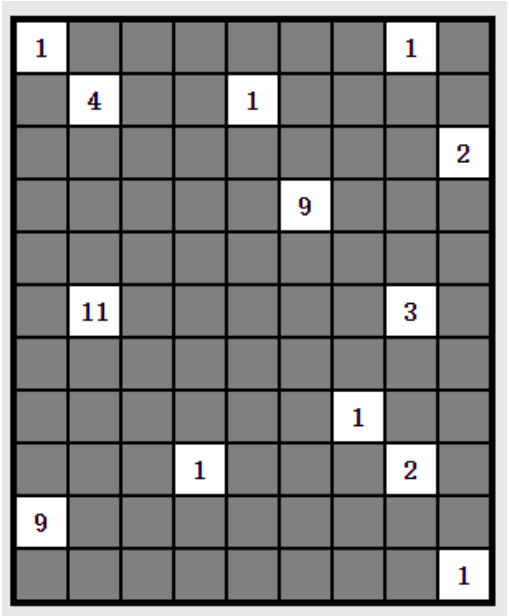
f) When solving large size of puzzles. 2 is a good number to solve first, because after we use rules like 2×2 or continuous ocean, etc. And then we can determine all oceans near the two cells.(we only need to determine one island and then we can get six or four oceans)

g) In a large size of puzzle, if there is a cell that no islands can reach to it, then it must be an ocean.

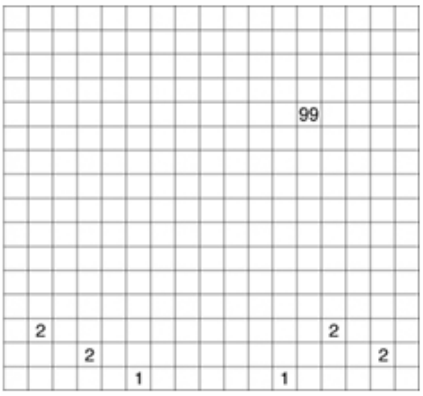
h) If an island of size N already has N-1 white cells identified, and there are only two remaining cells to choose from, and those two cells touch at their corners, then the cell between those two that is on the far side of the island must be black.(This is a very strong condition but this is also a common case)

There are so many strategies that can be found when solving a Nurikabe puzzle and it is not necessary to list all of them, because strategies can only solve parts of the puzzle. Strategies are more likely to solve puzzles with a small size and using small numbers. There are some puzzles which are difficult to do by hand.

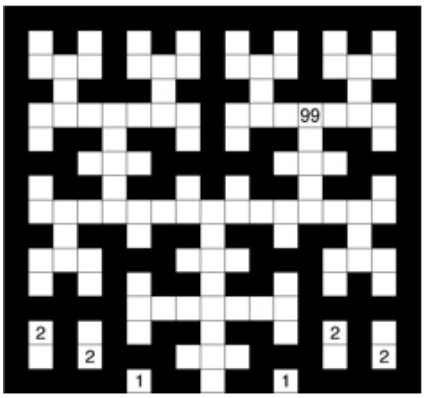
This puzzle is an example of one that is not easy to solve.



Part of this puzzle can be solved easily, but not for 11, 4 and 9.
Another example:



And the unique answer is



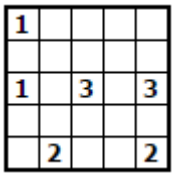
Those two examples show that some puzzles are very difficult to solve by logic, so we need find another way to solve them.

4. THREE WAYS OF PROGRAMING

There are so many different way to program with different languages and I will select three ways. The first way is designed by myself.

1. The main idea use the matrix to solve this problem. The basic idea is very simple. Use 1 to present the island, use 0 to present the ocean in the output matrix. Use A(i,j) to represents the element at the i row and j column of the matrix. So use this matrix as the input matrix. And call the input matrix as matrix A.

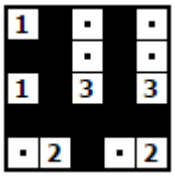
For example



For this puzzle, the matrix A should be

1 0 0 0 0
0 0 0 0 0
1 0 3 0 3
0 0 0 0 0
0 2 0 0 2

When input this matrix, through the computer program, output a matrix which all elements are 1 or 0 and which is the answer of this puzzle. And we call the output matrix as matrix B.



The matrix B should be

1 0 1 0 1
0 0 1 0 1
1 0 1 0 1
0 0 0 0 0
1 1 0 1 1

In order to solve the problem, I need to create all the possible matrixes first.
I use the R language here.

X = expand.grid(c(0,1), c(0,1), c(0,1), c(0,1), c(0,1), c(0,1), c(0,1), c(0,1),c(0,1), c(0,1), c(0,1), c(0,1), c(0,1), c(0,1), c(0,1), c(0,1), c(0,1), c(0,1), c(0,1), c(0,1), c(0,1), c(0,1), c(0,1), c(0,1),c(0,1))
rownames(X) = paste0('Matrix', 1:(225))**

Matrices = lapply(split(X, rownames(X)), matrix, 5, 5)

As we can see, there are 2^25 matrixes and most of them are impossible to be the right answer. So we need to filter the possible right answers from 2^25 matrixes. First, filter the matrix that B(i, j)+ B(i, j+1)+ B(i+1, j)+ B(i+1, j+1)>0 for any i and j from 1 to 3. So that the matrixes can match the condition that no 2 X 2 elements in the matrix are ocean.

```
for(i in 1: 33554432)
{ for (a in 1:5)
{for(b in 1:5)
{if
(Matrices$Matrixi[a,b]+Matrices$Matrixi[a+1,b]+Matrices$Matrixi[a,b+1]+
Matrices$Matrixi[a+1,b+1]=0)
print(i)}}}
```

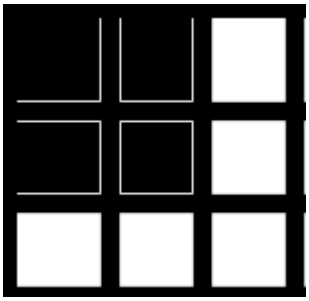
And those matrixes are all impossible because there exist 2×2 oceans.

Then talk about how many matrixes exist after the no 2×2 rule.

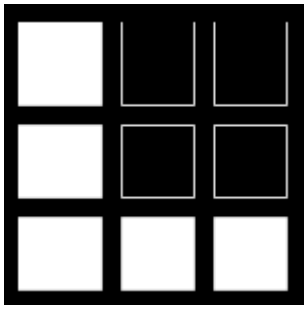
Theorem 1: when the size is three, there are 417 cases when there is no oceans. Use the Law of Inclusion and Exclusion to prove it.

$$\left|\bigcup_{i=1}^{n} A_i\right| = \sum_{i=1}^{n} |A_i| - \sum_{1 \le i < j \le n} |A_i \cap A_j| + \sum_{1 \le i < j < k \le n} |A_i \cap A_j \cap A_k| - \ldots + (-1)^{n-1} |A_1 \cap \cdots \cap A_n|.$$

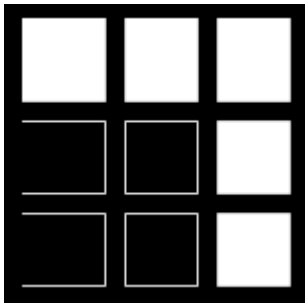
A1: The top left corner of 2×2 matrix is ocean



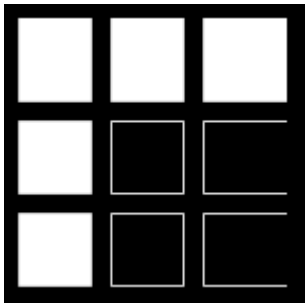
A2: The top right corner of 2×2 matrix is ocean



A3: The left bottom corner of 2×2 matrix is ocean.



A4: The right bottom corner of 2×2 matrix is ocean.



Then use the 2^5×4-2^3×4-2^2×2+2×4-1=95 to get 2^9-95=417

□

Next, we need to filter the matrix that the each piece of oceans connected to each other. We can build this rule by saying that if B(i,j)=0, then B(i,j-1)+B(i+1,j)+B(i,j+1)+B(i-1,j)<4.

We need to use the matrixes which are filtered, not all matrixes.

for(j in i)
{ for (a in 1:5)
{for(b in 1:5)
{if
(Matrices$Matrixi[a,b]=0,
Matrices$Matrixi[a,b-1]+Matrices$Matrixi[a+1,b]+Matrices$Matrixi[a,b+1]+Matrices$Matrixi[a-1,b]>0)
print(i) }}}

Then, i form 1 to 5, j from 1 to 5. If A(i,j)>0, then B(i,j)=1

Then we need to build a new various n equal to 1.And use the Conditional Structures. For example, A(3,3)=3, we need to judge if B(3,2),B(3,4),B(2,3) and B(4,3) equal to 0. If B(3,2) is not equal to 0, then make n equal to n+1 and continue the loop structure to judge if B(2,2),B(4,2)and B(3,1) equal to 0. If they are equal to0, then the loop structure will stop and we need to filter the matrix that n equal to 3.

Through this step, we will get a number i, and then we use

Matrices(i)

That is the answer matrix.

The advantage of this program is that its principle is very simple and it can solve all kinds of puzzles. The disadvantage of this program is that it will cost lots of

time and room to store so many matrixes. This will be discusses later.

2. The second way to solve this problem is by logic.
   This way also use the same rules as the way one, but it does not create matrix to solve it. It use rules and make hypothesis and use logic to get the answer step by step. The advantage of this way is that it can solve this problem without creating all the possible solutions. The disadvantage of it is that the programing will be very complex and it does work out for all possible Nurikabe puzzle, which means a small number of Nurikabe puzzle cannot be solved.
   The code is very long, and I only show parts of it here.

```
static int level5contradiction(int lev) {
   int z,r,l,k,oldsp=getstackpos();
   static int i=0,j=0;
     if(i>=x) i=0; if(j>=y) j=0;
   for(z=0;z<x*y;z++) {
      if(m[i][j]==UNFILLED) {
         /*    assume wall */
         copyboard(m,lev5bak);
         m[i][j]=BLOCKED;
         updatetoscreen(i,j,0);
         r=dogreedy(lev);
         if(r<0) {
            /*    contradiction! */
            copyboard(lev5bak,m);
            recalcboard();
            setstackpos(oldsp);
            addmovetoqueue(i,j,EMPTY);
            return 1;
         }
         copyboard(m,lev5alt1);
         copyboard(lev5bak,m);
         recalcboard();
         setstackpos(oldsp);
         /*    assume blank */
         m[i][j]=EMPTY;
         recalccompleteness(i,j,0);    /*    update st[][] */
         cleanupbfs();
         for(k=0;k<x;k++) for(l=0;l<y;l++) if(visit[k][l]) logprintf("visit
not removed at %d,%d\n",k,l);
         r=dogreedy(lev);
         if(r<0) {
            /*    contradiction! */
```

```
            copyboard(lev5bak,m);
            recalcboard();
            setstackpos(oldsp);
            addmovetoqueue(i,j,BLOCKED);
            return 1;
          }
          copyboard(m,lev5alt2);
          copyboard(lev5bak,m);
          recalcboard();
          setstackpos(oldsp);
          for(r=k=0;k<x;k++) for(l=0;l<y;l++)
if(lev5alt1[k][l]!=UNFILLED && lev5alt1[k][l]==lev5alt2[k][l] &&
m[k][l]==UNFILLED)
            addmovetoqueue(k,l,lev5alt1[k][l]),r=1;
          if(r) return 1;
        }
        j++;
        if(j==y) {
          j=0,i++;
          if(i==x) i=0;
        }
      }
      return 0;
}
```

This code means: Assume that a cell is white, fill greedily.
Assume that the same cell is black, fill greedily.
If one of the assumptions results in a contradiction,
Accept the other initial assumption.
If no contradiction, accept all cells that were filled in similarly.
One of the differences between Nurikabe solved by human and by computer is that computer can make hypothesis and test it easily, but human cannot.

And another example:

```
      if(!reach[i][j]) continue;
      for(k=0;k<4;k++) {
        x2=i+dx[k]; y2=j+dy[k];
        if(x2<0 || y2<0 || x2>=x || y2>=y || m[x2][y2]<EMPTY) continue;
        if(numreach[islandmap[x2][y2]]==1) {
          addmovetoqueue(i,j,EMPTY);
          ok=1;
        }
      }
    }
    return ok;
```

```
}

static int level4hint() {
   if(level4articulationpointblocked()) return 1;
   if(level4illegalgrow()) return 1;
   if(level4advancedarticulationisland()) return 1;
   if(level4islandmatchseparate()) return 1;
   if(level4connectborder()) return 1;
   return 0;
}
```

This is a one of the simple rules. It means if an unfilled cell is next to a cell which is reachable in only one way, it must be empty.
There are so many similar codes that uses some simple rules just like this one.
This code is about two thousand lines. The complete code can be found at https://github.com/stubbscroll/PUZZLE/blob/master/nurikabe.c

3. The third way and the second way have the same idea. There is a video about this solution.
   https://youtu.be/qyecvp2EqgA
   Both of those two programs use the basic rules to solve parts of the problem first. Then the computer use the hypothesis to solve the rest. And then use both of them again and again. The hypothesis process is trying to find out how many possible islands that can be connected by one cell and the program will test every cell and compare it with the number on the cell. If every cell's number is smaller than the number of possible islands it can connected, then the hypothesis is right. Otherwise is false.

5. HOW MANY NURIKABE ANSWERS EXIST?

Next, find how many possible Nurikabe answers in a fixed size. This questions seems to be complex because the answer of the Nurikabe puzzles need to satisfy all the rules. (Note: The interest is how many Nurikabe answers exist, not how many Nurikabe puzzles exist)
First, find the maximum and minimum of islands on a fixed size.
The maximum should be the NxN,

N	0	1	2	3	4	5	6	7	8	9
max	0	1	4	9	16	25	36	49	64	81

This means all the cells are islands.

And the minimum of islands should be

N	0	1	2	3	4	5	6	7	8	9
min	0	0	1	1	4	4	9	9	16	16

The strategy of find the minimum number is trying to make full use of a single island, which means use an island to make as many as passible oceans not form 2×2 cells.

Theorem2: When the size N is odd, the minimum of the islands is $((N-1)/2)^2$.
When N is even, the minimum of the islands is $(N/2)^2$.
Proof:
This number is the same as the maximum number of 2×2 tiles that fit on an n ×n board. (When we get the maximum number of 2×2 tiles on an N×N board, which means that we cannot find any more 2×2 cells that do not have tiles on them. Because if we can find another 2×2 cell to put another tile, then the number won't be the maximum)
Next calculate the maximum number of 2×2 tiles that fit on an n× n board in a binary fashion.
When N is even, the maximum case is to cover the whole board and the number of tiles are $(N/2)^2$
When N is odd, the length and the width of a tail is 2, which is even. If there are x tiles one row, it will be N-2x lists that are not covered by the tail. Finally, at least one list and one row of the board are not covered by tails. So the number of tiles are $((N-1)/2)^2$

□

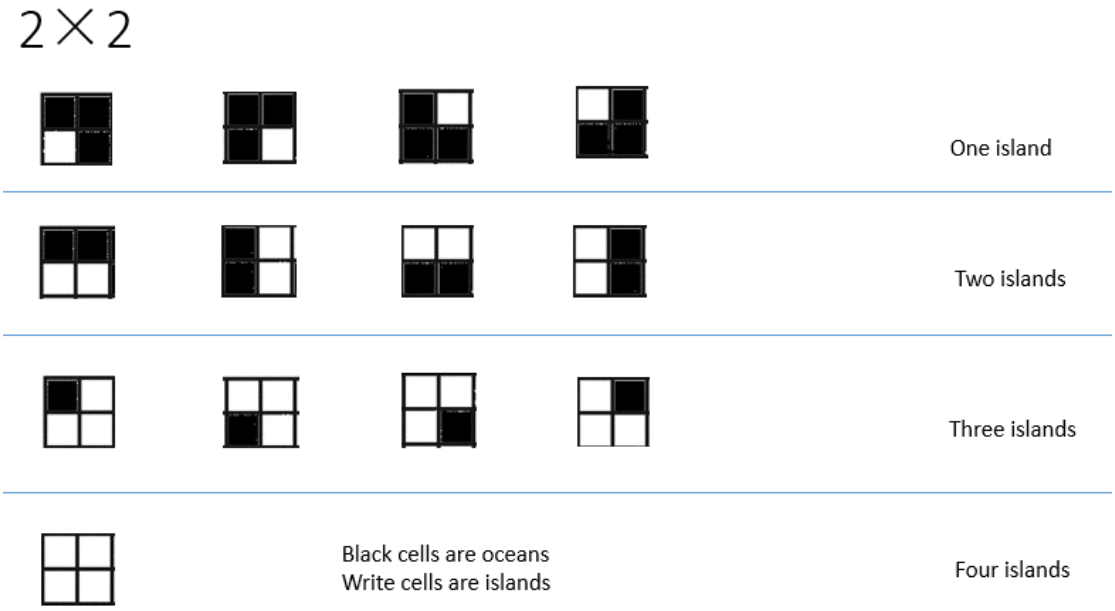
In order to find how many possible Nurikabe puzzles exist when the size is fixed, we need to consider the number of islands we use first and calculate them separately. The basic idea of this solution is that we calculate it with n islands, and then we add one island to get the number of solutions with n+1 islands. Most of this process is based on the formal results.

1. For example, let's start with a size 1×1 puzzle.

There are only two cases and both of them fit the rules.

2. When the puzzle is 2×2.
   a) When there is only one island, there are four cases.
   b) When there are two islands, there are four cases. Those four cases are created by adding another island to the first case.
   c) When there are three islands, there are four cases. Those four cases are created by adding another island to the second case.

d) When there are four islands, there are one cases.



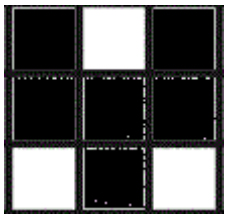
In conclusion, there are thirteen cases in total.

3. When the puzzle is 3×3.
   a) When there is only one island, there is only one case.
   b) When there are two islands, there are ten cases (10=8+2). We can use the result from the former case, and we need to put the other island anywhere else, which are eight cases. There are two more cases:


Plot 1


Plot2

   c) When there are three islands, there are 20 cases (20=8+8+4). Based on the last case, there are eight different cases, which are gained by adding another island next to the existing islands. We can get another eight cases by adding one island into plot 1 and plot 2.
   There are another four cases like plot 3


Plot 3

When the number of islands are greater than two, there will not be any islands

that connected diagonally. Because if there is any islands that connected diagonally, there will be isolate ocean. So this means that when islands are connected, they must be connected left/right and up/down. When we know this, the counting will be easier.

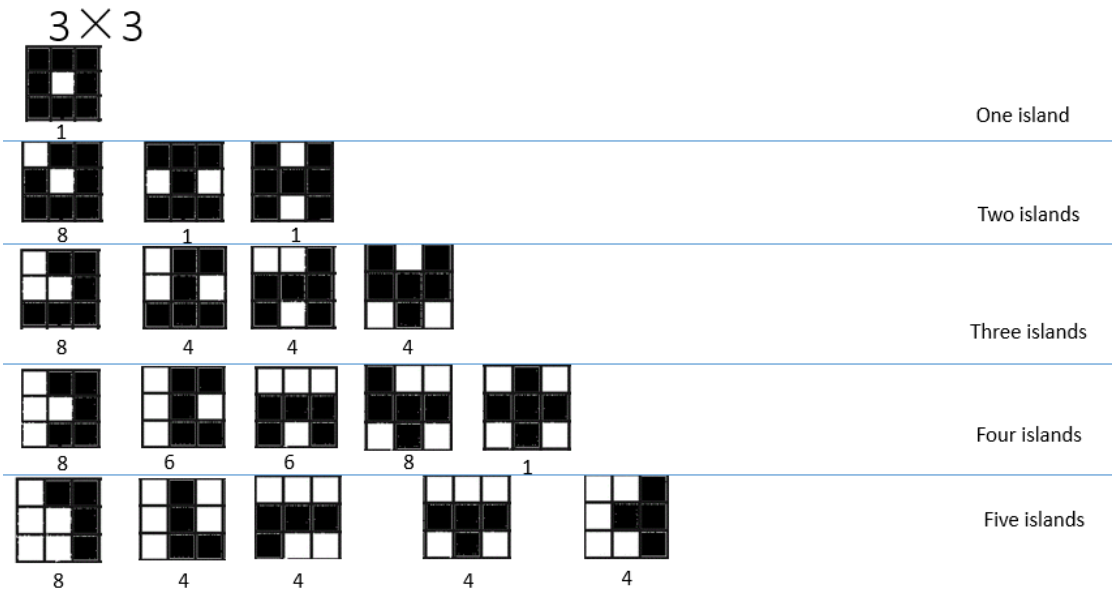
d) When there are four islands, there are 29 cases (29=8+12+8+1). The first eight can be get from the same way, the twelve can be get from 2×6, and we can also get eight form the formal case. The last one is new like this:

Plot 4

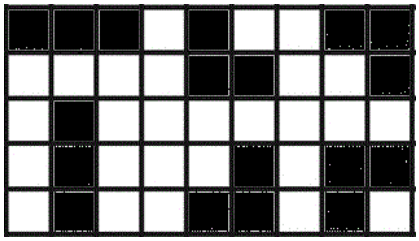
e) When there are five islands, there are 28 cases (28=8+4+4+4+4), and we can get 8+4+4+4 form the formal result, and add a new case:

Plot 5

f) When the number of island is greater than 6, it is not necessary to use this method, because most of cases are come to one case. At this time, the number of island is large and the number of oceans is small. So we also can place the oceans in different places. Find the potential ocean is straightforward.

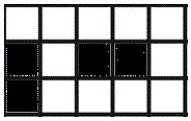


When there are six islands, there will be three oceans. All the possible form of the oceans are:



So there will be 22=3+3+4×4

When the number of islands is seven, there will be only two oceans. All the possible form of the oceans are:



So there are 132 cases in total.

And then, calculate the proportion of Nurikabe puzzle answers in all possible cells.

Size	1	2	3
Number of Nurikabe	2	13	132
Totally number	2	16	512
proportion	1	0.8125	0.2578

The proportion is decreasing very quickly, which means that when N is greater, most of the all possible cases are not Nurikabe puzzle answers. Because the number of

A239748 Number of distinct sequences defined by the elements of powers >= 0 of n X n (0, 1) matrices.

2, 13, 132, 3833, 363288 (list; graph; refs; listen; history; text; internal format)

OFFSET 1, 1

LINKS Table of n, a(n) for n=1..5.
Christopher Hunt Gribble, C++ Program

EXAMPLE a(2) = 13 because there are 13 distinct sequences in the set of 256 sequences formed by each element of each 2 X 2 binary matrix raised to successive powers >= 0. The first 10 terms of the distinct sequences and the frequencies of occurrence are:

Sequence	Frequency
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...	16
0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...	2
0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, ...	2
0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, ...	4
0, 1, 1, 2, 3, 5, 8, 13, 21, 34, ...	4
0, 1, 2, 3, 4, 5, 6, 7, 8, 9, ...	2
0, 1, 2, 4, 8, 16, 32, 64, 128, 256, ...	2
1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...	12
1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, ...	2
1, 0, 1, 1, 2, 3, 5, 8, 13, 21, ...	2
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, ...	12
1, 1, 2, 3, 5, 8, 13, 21, 34, 55, ...	2
1, 1, 2, 4, 8, 16, 32, 64, 128, 256, ...	2
.	Total 64

CROSSREFS Cf. A238596.
Sequence in context: A041509 A079165 A223075 * A065132 A047856 A246875
Adjacent sequences: A239745 A239746 A239747 * A239749 A239750 A239751

KEYWORD nonn, hard, more

AUTHOR Christopher Hunt Gribble, Mar 26 2014

STATUS approved

totally possible cases is $2^{N\wedge 2}$, which grows very fast.

The first few terms of the sequence is the same as the number of all possible Nurikabe puzzles in size N.

7. Open questions

a) Use the result from the former part 5 to fix the program 1. The disadvantage of program 1 is there are too many matrixes to create. The percentage of possible matrixes are smaller when n is large so design a new program to only create the useful matrixes.

b) Prove that the number series A235748 is the same as the number of all possible Nurikabe puzzles in size N.

There is a website to find lots of Nurikabe puzzle and it can check the result we have done.
http://www.puzzle-nurikabe.com/

In this final project, the program2, program 3 and the number series A235748 are found on the internet. All the other parts are made by myself.