Centered Hexagonal Number

Published by Matt in Java

formatting numbers strings

As stated on the On-Line Encyclopedia of Integer Sequences:

> The hexagonal lattice is the familiar 2-dimensional lattice in which each point has 6 neighbors.

A **centered hexagonal number** is a centered figurate number that represents a hexagon with a dot in the center and all other dots surrounding the center dot in a hexagonal lattice.

At the end of that web page the following illustration is shown:

```
Illustration of initial terms:
.
.                                  o o o o
.                      o o o      o o o o o
.          o o        o o o o     o o o o o o
.    o     o o o     o o o o o    o o o o o o o
.          o o        o o o o     o o o o o o
.                      o o o      o o o o o
.                                  o o o o
.
.    1       7          19             37
.
```

Write a function that takes an integer `n` and returns `"Invalid"` if `n` is not a **centered hexagonal number** or its illustration as a multiline rectangular string otherwise.

I am examining this challenge and few patterns have emerged:

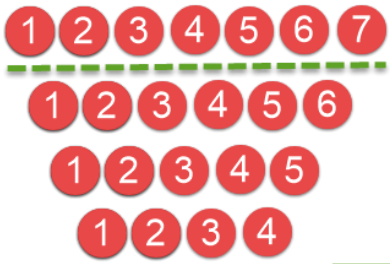
The middle dissection is odd multiple
There are n-1 rows (above or below) for each subsequent configuration where n=1 onwards

N=1	N=2	N=3	N=4	N=5
Middle dissection = 1 If (N==1) M=N	Middle dissection = 3 M = (N*2)-1	Middle dissection = 5 M=(N*2)-1	Middle dissection = 7 M=(N*2)-1	Middle dissection = 9 M=(N*2)-1
Number rows above or below is: R=N-1	Number rows above or below is: R=N-1	Number rows above or below is: R=N-1	Number rows above or below is: R=N-1	Number rows above or below is: R=N-1
H=1	H=7	H=19	H=37	H=61

I am seeing no relationship whatsoever between N and H.
I can attempt to use HashMap to store key value of “beads” in row.
This will provide a level of memorization to speed up execution when moving from N=1 to N=? (need to terminate when the value in H exceeds n (value passed into the function).

N=4

M=(N*2)-1=7



R=(N-1)=3

hashMp.put(6,6)
hashMP.put(5,5)
hashMP.put(4,4)

if (hashMP.containsKey(6))
It will use this sort of notation to interrogate the hashMap (as N increases).
It will readily get the value hashMP.get(6)
Alternatively hashmap can be discarded altogether since the value for row 1 = (M-1), row 2 = (M-2) and row 3 = (M-3)

Unfortunately the hashMap will not have the same advantages as factorials since hashMP.get(6) will not have 6 x 5 x 4 x 3 x 2 x 1

But I can contemplate a similar concept
in hashMP.put(6, (I will intend to store 6+5+4+3,+2,+1 = 21)
in hashMP.put(5, (I will intend to store 5+4+3,+2,+1 = 15)
in hashMP.put(4, (I will intend to store 4+3,+2,+1 = 10)
in hashMP.put(3, (I will intend to store 3,+2,+1 = 6)
in hashMP.put(1, (We know from the configurations that 1 can only appear in the dissection. There is not a valid hexagon which has 1 "bead" in any rows below or above.. Si I will simply store 0 here)

We know we will use 3 (R) rows:

So I am hoping hashMP.get(M-1) - hashMP.get(M-(R+1)
Will give the actual value of "beads" above the dissection:
hashMP.get(6) - hashMP.get(3) = This is equivalent to above highlighted in blue = 21 - 6 = 15

Now the total would be (15 x 2) + 7 = 37

Now just to verify if the memoization can be used.
The next N=5 M= 9
It would need to perform:

hashMP.containsKey(M-1) . It would not have the value for 8.
So it would need to calculate this and store value
hashMap.put(8, hashMap.get(7)+8)
However we can see that hashMap.get(7) has not been populated yet also..
.
So we know that each time N has increased by 1,
It has to perform:
hashMap.put(M-1, M-1) = hashMap.put(8, 8+hashMap.get(7)) = 36
hashMap.put(M-2, M-2) = hashMap.put(7, 7+hashMap.get(6)) = 28

Now hopefully it is ready to perform this calculation:
We know we will be using 4 rows

hashMP.get(M-1) - hashMP.get(M-(R+1)
hashMP.get(8) - hashMP.get(9-(4+1)
hashMP.get(8) - hashMP.get(4) = 36 - 10 = 26

Now the total would be (26 x 2) + 9 = 61

I am now going to present this hexagon to see if it is valid.

9 + 26 + 26 = 61

Examples

```
hexLattice(1) → " o "
// o

hexLattice(7) → "  o o  \n o o o \n  o o  "
//  o o
// o o o
//  o o

hexLattice(19) → "   o o o   \n  o o o o  \n o o o o o \n  o o o o  \n   o o o
//   o o o
//  o o o o
// o o o o o
//  o o o o
//   o o o

hexLattice(21) → "Invalid"
```

Notes

N/A