Magic Square Puzzle Board Validator

(report)

Fatima A

Introduction

The earliest known magic square is Chinese, recorded around 2800 B.C. It is a typical 3x3 magic square except that the numbers were represented by patterns not numerals. After that, in 1514, 4x4 magic square was know by in Albrecht Dürer's

Any way, this is focused as very interesting problem in math and lots of mathematics found out solving method of it. Today, this problem is often used as test problem for algorithm.

And then what is the magic square?

This is the matrix, of which each row, column, and diagonal in the square has the same sum. The sum is called magic constant. The formula of this is .

This is not only concept of the magic square but also the condition for solving mathematically. Today, the lots of method about this problem are known very well.

But we can clarify this two kind of method

The most brainless method is recursive method. The main point is the deep of stack. Because the order of magic square matrix is determined by this. Advantage is that it is very simple and we can find out all solution. But it can’t work at the big order of magic square matrix. It will takes lots of time and needs big capability of memory.

We can often see error message “stack overflow” from this.

The second method is one based on mathematics. In here, we can find general solving method for magic square. In the aspects over, it is much better than over method.

But it can find only one solution. In other word, one method respect to one solution.

In this report I’m going to write over this method.

In this task, I have to carry out some points.

1. The puzzle boards will be supplied to the program in the form of a simple text file.

2. The program should prompt the user for the filename of a text file to open – some Python error handling should be used in case the user enters the name of a file that does not exist.Because of many mathematical method there are many way for this.

3. The program should read the contents of the given text file and store this in a suitable Python data structure.

4. Appropriate validation should be in place to ensure the supplied file contains a complete n x n square grid of digits, and ensure the digits are all integers in the range 1 to ; a suitable error message should be returned where necessary.

5.Assuming a suitable n x n square grid of integers is successfully loaded into the chosen data structure, further checks should then be performed to ensure that the sums of each row, each column, and each of the two diagonals are all equal.

6.If any errors are detected in the puzzle board then details of these errors should be given to the user.

7.If the puzzle board passes all the validation checks, then it should be written into a new text file using the same format as the original input file (see the example text files above) – the name for this new text file should be the original filename entered by the user prefixed with “VALID\_”.

8.You should try to make your Python program as efficient as you can – consider segmenting my code into functions where appropriate for particular tasks.

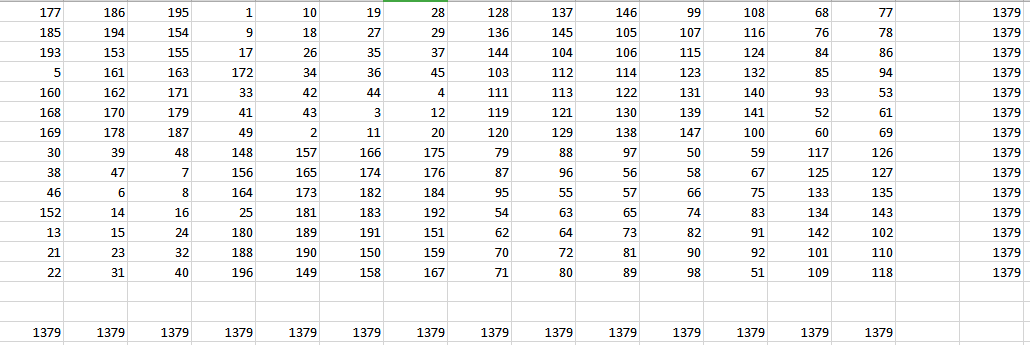


Figure. 14x14 magic square

1. Solution Explanation
   1. Design of Algorithm.

First, form mathematics principle, Algorithm is consist of there main part:

Odd order magic square.

Single even order magic square.

Double even order magic square.

There are different method in individual case.

* + 1. Solving an Odd order Magic Square

̶ Calculate the magic constant.



For example, n=9

c=369.

This constant will be used for checking result. The

̶ Place the number 1 in the center cell on the top row.

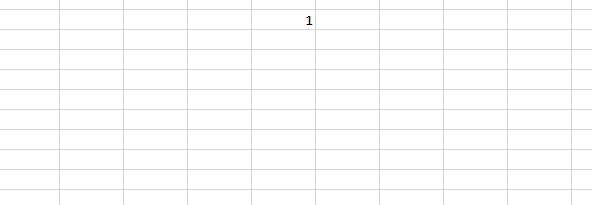


Figure 1-1. the position of the number 1

If odd order magic square is given, first number will place in the centre cell on the top row.

̶ Fill in the empty cell with other numbers using an up-one, right-one method.

Remain numbers will be filled sequentially by moving up one row, the one column to the right.

There are three case that we could have, at the time we have to apply following rules

First selecting the cell above the magic square’s top row, remain in that cell’s column, but place the number in the bottom row of that column.

Second selecting the cell to the right of the magic square’s right column, remain in that cell’s row, but put the number in the furthest left column of that row.

Third, selecting the cell that is already occupied, go back to the last cell that has been filled in, and place the next number directly below it.

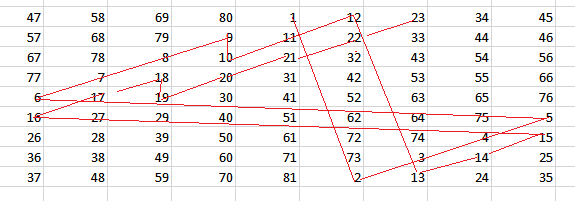


Figure 1-2. the rill for fill in the cell with numbers

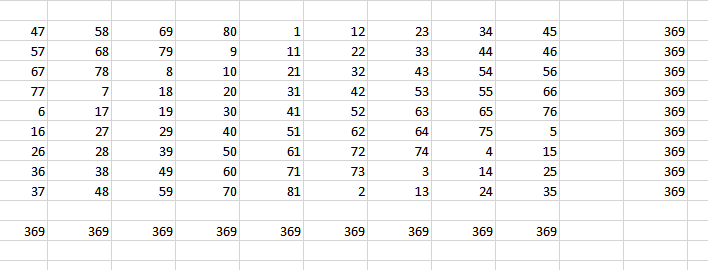


Figure 1-3.9x9 Odd order Magic Square

* + 1. Singly Even order Magic Square

It means the number which could be divided by 2 without remains but it is not possible option when it is divided by 4. for example 6, 14, 34....

̶ Divide the magic square

Divide the magic square into four child matrix of equal size.

Label them A (top left), C (top right), D (bottom left) and B (bottom right).

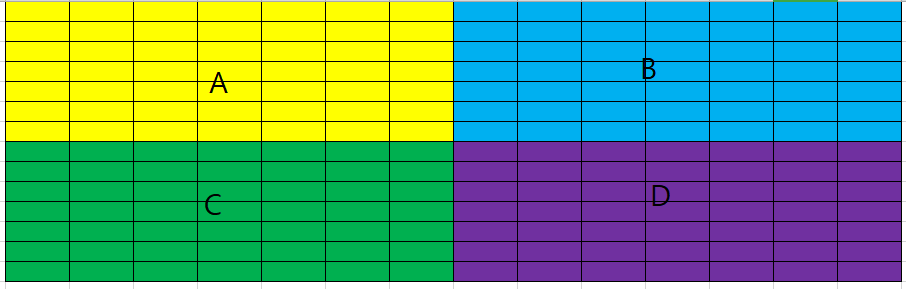


Figure 1-3.dividing to child matrices

̶ Place numbers in child matrix

The total numbers are divided into four part in order. For example, when the order is 6, it consists of 1-9, 10-18,19-27,28-36.

Make the child magic square with given child set of numbers. After single

even number is divided by 2, the results is odd number. So it is possible to use over odd algorithm for making child magic square.



Figure 1-4. child magic squares

̶ change some parts for making total magic square.

First, in the part A and D, some parts exchange each other.

In the child square A, select the central cell. And then the left parts from central cell(expect its row) in the A,D exchange except one cell. This cell

is the number in the left row, column of the central cell.

After that, the central cells exchange each other.



Figure 1-5. exchanging some parts

Second in the part B and C, some parts exchange each other.

The right parts from central cell(expect its row and next row) exchange each other.

After all done, it become the magic square.

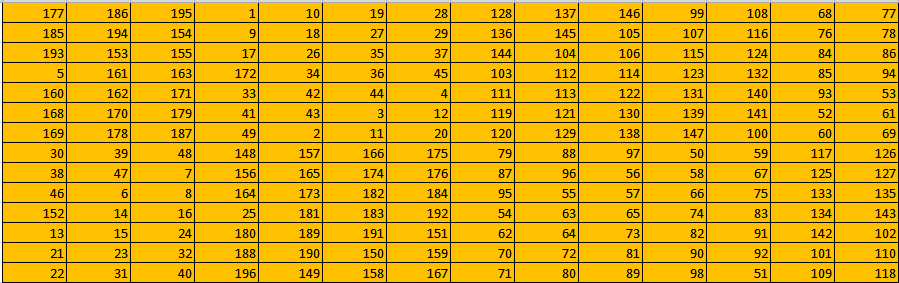


Figure 1-6. 14x14 Odd order Magic Square

* + 1. Doubly Even order Magic Square

̶ Divide nine child matrix of entire matrix.

The magic square is divided to nine parts like figure 1-7.

Fill in the all cell with numbers in order.

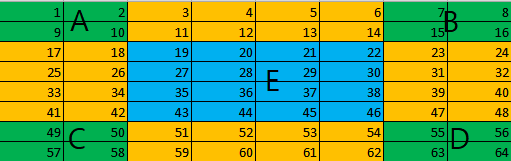


Figure 1-7. Dividing to part matrices

And then except part A,B,C and D, replace numbers in inverse order. At the time, original number is removed.

After all that it become the magic square.

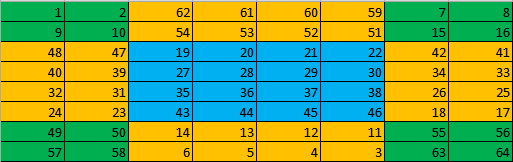


Figure 1-8. 8x8 Odd order Magic Square

So we can design the algorithm based on over principle.

* 1. Design of Algorithm.

̶ input the magic square order.

This value is inputted from text file. The title of the file is “input.txt”.

̶ check the validation of the input value.

In here, we have to check that the input value is integer. If it is not, prompt

user the error message.

̶ select module according to the type of input value.

As I just mentioned over, there are three module in here.

Therefor check the input value and then according to the result, select the

module.

̶ configure magic square matrix.

The magic square matrix is configured based on over rules.

̶ check the result.

If the result is correct, result is outputted to text file. The title is

“Validation\_NxN.txt”. If this is not, prompt the error message.

Figure... shows the total algorithm.

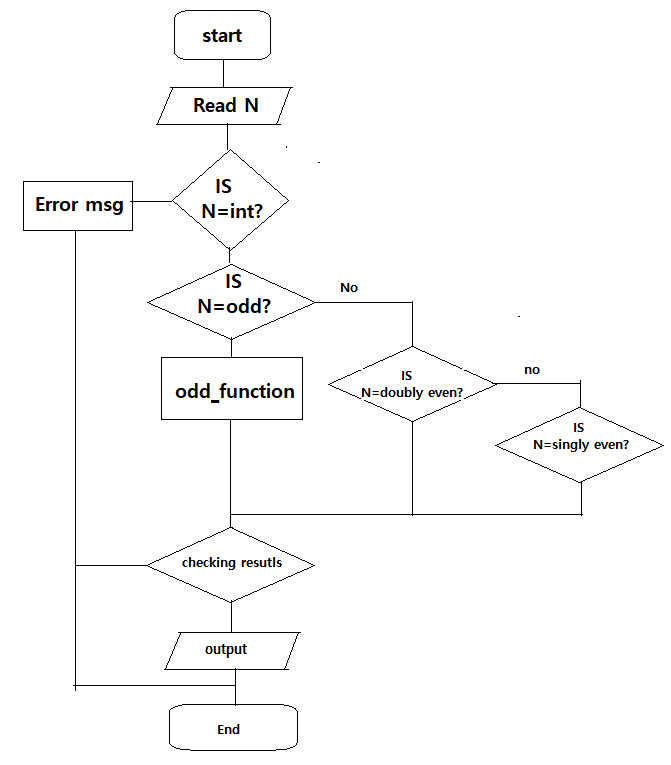


Figure 1-9. magic square algorithm

1. Test.

2.1 check the result.

̶ prepare the input material.

Input material is soon the magic square order. Input orders were prepared three

respectively.

Odd order:3,5,11,21,31

Singly even order:6, 14, 26

Doubly even order:4,8,16

̶ check the results.

This program had the checking module so, if there is no error message, it means

that the results are correct.

The results according to over input order are following.

ODD: 3,5,11

8 1 6

3 5 7

4 9 2

17 24 1 8 15

23 5 7 14 16

4 6 13 20 22

10 12 19 21 3

11 18 25 2 9

68 81 94 107 120 1 14 27 40 53 66

80 93 106 119 11 13 26 39 52 65 67

92 105 118 10 12 25 38 51 64 77 79

104 117 9 22 24 37 50 63 76 78 91

116 8 21 23 36 49 62 75 88 90 103

7 20 33 35 48 61 74 87 89 102 115

19 32 34 47 60 73 86 99 101 114 6

31 44 46 59 72 85 98 100 113 5 18

43 45 58 71 84 97 110 112 4 17 30

55 57 70 83 96 109 111 3 16 29 42

56 69 82 95 108 121 2 15 28 41 54

Singly even:6,14,18

35 1 6 26 19 24

3 32 7 21 23 25

31 9 2 22 27 20

8 28 33 17 10 15

30 5 34 12 14 16

4 36 29 13 18 11

177 186 195 1 10 19 28 128 137 146 99 108 68 77

185 194 154 9 18 27 29 136 145 105 107 116 76 78

193 153 155 17 26 35 37 144 104 106 115 124 84 86

5 161 163 172 34 36 45 103 112 114 123 132 85 94

160 162 171 33 42 44 4 111 113 122 131 140 93 53

168 170 179 41 43 3 12 119 121 130 139 141 52 61

169 178 187 49 2 11 20 120 129 138 147 100 60 69

30 39 48 148 157 166 175 79 88 97 50 59 117 126

38 47 7 156 165 174 176 87 96 56 58 67 125 127

46 6 8 164 173 182 184 95 55 57 66 75 133 135

152 14 16 25 181 183 192 54 63 65 74 83 134 143

13 15 24 180 189 191 151 62 64 73 82 91 142 102

21 23 32 188 190 150 159 70 72 81 90 92 101 110

22 31 40 196 149 158 167 71 80 89 98 51 109 118

290 301 312 323 1 12 23 34 45 209 220 231 242 163 174 104 115 126

300 311 322 252 11 22 33 44 46 219 230 241 171 173 184 114 125 127

310 321 251 253 21 32 43 54 56 229 240 170 172 183 194 124 135 137

320 250 261 263 31 42 53 55 66 239 169 180 182 193 204 134 136 147

6 260 262 273 284 52 63 65 76 168 179 181 192 203 214 144 146 157

259 270 272 283 51 62 64 75 5 178 189 191 202 213 224 145 156 86

269 271 282 293 61 72 74 4 15 188 190 201 212 223 234 155 85 96

279 281 292 303 71 73 3 14 25 198 200 211 222 233 235 84 95 106

280 291 302 313 81 2 13 24 35 199 210 221 232 243 164 94 105 116

47 58 69 80 244 255 266 277 288 128 139 150 161 82 93 185 196 207

57 68 79 9 254 265 276 287 289 138 149 160 90 92 103 195 206 208

67 78 8 10 264 275 286 297 299 148 159 89 91 102 113 205 216 218

77 7 18 20 274 285 296 298 309 158 88 99 101 112 123 215 217 228

249 17 19 30 41 295 306 308 319 87 98 100 111 122 133 225 227 238

16 27 29 40 294 305 307 318 248 97 108 110 121 132 143 226 237 167

26 28 39 50 304 315 317 247 258 107 109 120 131 142 153 236 166 177

36 38 49 60 314 316 246 257 268 117 119 130 141 152 154 165 176 187

37 48 59 70 324 245 256 267 278 118 129 140 151 162 83 175 186 197

Doubly even:4,8,16

1 15 14 4

12 6 7 9

8 10 11 5

13 3 2 16

1 2 62 61 60 59 7 8

9 10 54 53 52 51 15 16

48 47 19 20 21 22 42 41

40 39 27 28 29 30 34 33

32 31 35 36 37 38 26 25

24 23 43 44 45 46 18 17

49 50 14 13 12 11 55 56

57 58 6 5 4 3 63 64

1 2 3 4 252 251 250 249 248 247 246 245 13 14 15 16

17 18 19 20 236 235 234 233 232 231 230 229 29 30 31 32

33 34 35 36 220 219 218 217 216 215 214 213 45 46 47 48

49 50 51 52 204 203 202 201 200 199 198 197 61 62 63 64

192 191 190 189 69 70 71 72 73 74 75 76 180 179 178 177

176 175 174 173 85 86 87 88 89 90 91 92 164 163 162 161

160 159 158 157 101 102 103 104 105 106 107 108 148 147 146 145

144 143 142 141 117 118 119 120 121 122 123 124 132 131 130 129

128 127 126 125 133 134 135 136 137 138 139 140 116 115 114 113

112 111 110 109 149 150 151 152 153 154 155 156 100 99 98 97

96 95 94 93 165 166 167 168 169 170 171 172 84 83 82 81

80 79 78 77 181 182 183 184 185 186 187 188 68 67 66 65

193 194 195 196 60 59 58 57 56 55 54 53 205 206 207 208

209 210 211 212 44 43 42 41 40 39 38 37 221 222 223 224

225 226 227 228 28 27 26 25 24 23 22 21 237 238 239 240

241 242 243 244 12 11 10 9 8 7 6 5 253 254 255 256

2.2 check exceptions.

̶ run the program without input text file.

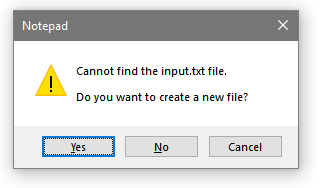


Figure 2-1. Error message to create new input file

̶ input value which is no integer for example string, float.

Input:”test”,”32.23”

Reust:

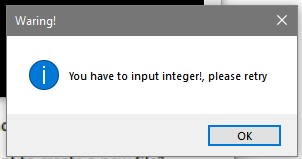


Figure 2-2. Error message to detect invalid value.

1. Conclusion.

I think, it was good for me to program the magic square. Program works well as

mentioned above the requirement.

But this did not include only one method for magic square so we can’t get all

solutions in possible cases. This issue can be solved by recursive method.

In future , I’m going to add this module so, it will complete more perfectly.

1. Appendix.

import numpy as np

import subprocess

from tkinter import messagebox

#checking order

def check\_order(N):

if N%2==1:

odd\_magic\_square=odd\_even\_square(0,N)

output\_file(odd\_magic\_square)

if N%4==0:

doubly\_magic\_square(N)

if N%4!=0 and N%2==0:

singley\_magic\_square(N)

#doubly even magic square function

def doubly\_magic\_square(N):

magic\_square = np.zeros((N,N), dtype=int)

magic\_flage=np.zeros((N,N), dtype=int)

i=j=0;k1=1;k2=N\*\*2;sm\_order=int(N/4)

for i in range(0,N):

for j in range(0, N):

magic\_square[i,j]=k1;k1+=1;

if i<sm\_order or i>N-sm\_order-1:

if j>sm\_order-1 and j<N-sm\_order:

magic\_square[i,j]=k2

if i>sm\_order-1 and i<N-sm\_order:

if j<sm\_order or j>=N-sm\_order:

magic\_square[i,j]=k2

k2=k2-1

output\_file(magic\_square)

#odd even magic square function

def odd\_even\_square(start\_number,order):

magic\_square = np.zeros((order,order), dtype=int)

n = 1

i, j = 0, order//2

while n <= order\*\*2:

magic\_square[i, j] = n+start\_number

n += 1

newi, newj = (i-1) % order, (j+1)% order

if magic\_square[newi, newj]:

i += 1

else:

i, j = newi, newj

return magic\_square

#singley even magic squre function

def singley\_magic\_square(N):

#calculating the order of sub matrix

sm\_order=int(N/2)

#structuring sub magic\_square matrix

MA=odd\_even\_square(0,sm\_order)

s\_n=sm\_order\*\*2

MB=odd\_even\_square(s\_n,sm\_order)

s\_n=2\*sm\_order\*\*2

MC=odd\_even\_square(s\_n,sm\_order)

s\_n=3\*sm\_order\*\*2

MD=odd\_even\_square(s\_n,sm\_order)

ex\_n=int(sm\_order/2)-1

for j in range(sm\_order-ex\_n,sm\_order):

for i in range(0,sm\_order):

temp=MC[i,j];MC[i,j]=MB[i,j];MB[i,j]=temp

for j in range(0,ex\_n+1):

for i in range(0,sm\_order):

if i==ex\_n+1 and j==0:

continue

temp=MA[i,j];MA[i,j]=MD[i,j];MD[i,j]=temp

temp=MA[ex\_n+1,ex\_n+1];MA[ex\_n+1,ex\_n+1]=MD[ex\_n+1,ex\_n+1];MD[ex\_n+1,ex\_n+1]=temp

singley\_magic\_square=np.zeros((N,N),dtype=int)

#construct total magic\_square matrix.

for i in range(0,N):

for j in range(0,N):

if i<sm\_order and j<sm\_order:

singley\_magic\_square[i,j]=MA[i,j]

if i>=sm\_order and j<sm\_order:

no1=i-sm\_order

singley\_magic\_square[i,j]=MD[no1,j]

if i<sm\_order and j>=sm\_order:

no2=j-sm\_order

singley\_magic\_square[i,j]=MC[i,no2]

if i>=sm\_order and j>=sm\_order:

no1=i-sm\_order;no2=j-sm\_order

singley\_magic\_square[i,j]=MB[no1,no2]

output\_file(singley\_magic\_square)

#output functon

def output\_file(result):

filename=str(N)+'×'+str(N)+".txt"

np.savetxt(filename,result, fmt="%5d")

subprocess.Popen(["notepad",filename])

check\_result(result)

def check\_result(result):

s1=0;magic\_number=N\*(N\*\*2+1)/2

for i in range(0,N):

s1=0

for j in range(0,N):

s1+=result[i,j]

if s1!=magic\_number:

messagebox.showinfo("Waring","This is no magic square!")

return 0

filename="Valid\_output"+str(N)+'×'+str(N)+".txt"

np.savetxt(filename,result, fmt="%5g")

#starting program

def main():

print("Enter order:")

inp=subprocess.Popen(["notepad","input.txt"])

global N

while inp.wait():

pp+=1

f=open("input.txt","r")

content=(f.read())

#check validation of input.

try:

N=int(content)

except Exception as e:

messagebox.showinfo("Waring!","You have to input integer!, please retry")

return 0

if (float(content)-int(float(content)))!=0:

messagebox.showinfo("Waring!","You have to input integer!, please retry")

return 0

N=int(content)

if N<2:

messagebox.showinfo("Waring!","input error!, please retry")

return 0

#select method

check\_order(N)

if \_\_name\_\_== "\_\_main\_\_":

main()