light_out

December 27, 2021

1 Generate Matrix

general method to generate a square matrix of square game

```
[1]: import numpy as np
     def genenerate_neighbord_matrix(n) -> np.array:
         mat = np.zeros((n**2, n**2), dtype= np.int8)

         # the general case
         for j in range(0, n**2):
             if j-n > -1 :
                 mat[j-n,j] = 1

             if j % n != 0 :
                 mat[j-1,j] = 1

             mat[j,j] = 1

             if (j+1) % n != 0 :
                 mat[j+1,j] = 1

             if j+n < n**2 :
                 mat[j+n,j] = 1

         return mat
     print(genenerate_neighbord_matrix(3))
```

```
[[1 1 0 1 0 0 0 0 0]
 [1 1 1 0 1 0 0 0 0]
 [0 1 1 0 0 1 0 0 0]
 [1 0 0 1 1 0 1 0 0]
 [0 1 0 1 1 1 0 1 0]
 [0 0 1 0 1 1 0 0 1]
 [0 0 0 1 0 0 1 1 0]
 [0 0 0 0 1 0 1 1 1]
 [0 0 0 0 0 1 0 1 1]]
```

2 Solving game

general method to how solve the game, by solving the matrix.

```
[2]: from sage.all import *
     n = 3
     A = Matrix(Integers(2),genenerate_neighbord_matrix(n))
     Y = vector([1 for x in range(n**2)])
     Z = vector([0 for x in range(n**2)])
     X = A.solve_right(Y)
     print(X)
```

```
(1, 0, 1, 0, 1, 0, 1, 0, 1)
```

3 Spanish method

```
[3]: def gaussian_elimination_spanish_alg(mat : np.array, sol_vec :np.array):
         n = int(sqrt(mat.shape[0]))
         #all rows but the last one
         for i in range(0, n**2-n):
             # the lamp that is affected
             affected_lamp = i + n
             row_i = mat[i][:affected_lamp+1]
             # check rows below
             # for j in range(i+1, n**2):
             for j in [i-1 + n, i+n, i+n+1, i+ 2*n]:
                 if j> -1 and j < n**2 and mat[j][affected_lamp] == 1:
                     row_j = mat[j][:affected_lamp+1]
                     row_j = row_j + row_i
                     row_j = row_j % 2
                     mat[j][:affected_lamp+1] = row_j
                     sol_vec[j] = ( sol_vec[j] + sol_vec[i] ) % 2


     def mul_mat_sol_based_on_res(mat : np.array, end_state : list, res : list):
         n = int(sqrt(mat.shape[0]))
         for i in range(0,n**2-n):
             res_i_plus_n = int(end_state[i])
             for j in range(0,i+n):
                 res_i_plus_n = (res_i_plus_n + mat[i][j] * res[j]) % 2
             res.append(res_i_plus_n)

     def generate_mat_spanish_alg(mat : np.array):
         n = int(sqrt(mat.shape[0]))
         end_state = np.ones(n**2)
         gaussian_elimination_spanish_alg(mat, end_state)
         # the matrix we need to solve
```

```
        new_mat = np.array(mat[n**2-n:n**2, 0:n], copy=True)
        new_sol = np.array(end_state[n**2-n:n**2],  copy=True)

        #find solution for n variables
        A = Matrix(Integers(2),new_mat)
        Y = vector(Integers(2),new_sol)
        X = A.solve_right(Y)
        res = [x for x in X]
        mul_mat_sol_based_on_res(mat, end_state, res)
        return res

mat = genenerate_neighbord_matrix(4)
A = Matrix(Integers(2),mat)
res = generate_mat_spanish_alg(mat)
print(mat)
print(res)

print('check solution:')
X = vector(Integers(2),res)
Y = A*X
print(Y)
```

```
[[1 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0]
 [1 1 1 0 0 1 0 0 0 0 0 0 0 0 0 0]
 [0 1 1 1 0 0 1 0 0 0 0 0 0 0 0 0]
 [0 0 1 1 0 0 0 1 0 0 0 0 0 0 0 0]
 [1 0 1 0 0 0 0 0 1 0 0 0 0 0 0 0]
 [0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 0]
 [1 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0]
 [0 1 0 1 0 0 0 0 0 0 0 1 0 0 0 0]
 [0 1 1 1 0 0 0 0 0 0 0 0 1 0 0 0]
 [1 1 0 1 0 0 0 0 0 0 0 0 0 1 0 0]
 [1 0 1 1 0 0 0 0 0 0 0 0 0 0 1 0]
 [1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 1]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]]
[0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1]
check solution:
(1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1)
```

4   Minimal case

generate matrix for rectengle game.
searching for integer solution.

```
[4]: import numpy as np

     # to prove the minimal case on not square we need to build matrix for not␣
      ↪rectangler board
     def genenerate_neighbord_matrix_m_n(m,n) -> np.array:
         mat = np.zeros((m*n, m*n), dtype= np.int8)

         # the general case
         for j in range(0, m*n):
             if j-n > -1 :
                 mat[j-n,j] = 1

             if j % n != 0 :
                 mat[j-1,j] = 1

             mat[j,j] = 1

             if (j+1) % n != 0 :
                 mat[j+1,j] = 1

             if j+n < n**2 :
                 mat[j+n,j] = 1

         return mat
     print(genenerate_neighbord_matrix_m_n(3,2))
```

```
[[1 1 1 0 0 0]
 [1 1 0 1 0 0]
 [1 0 1 1 1 0]
 [0 1 1 1 0 1]
 [0 0 0 0 1 1]
 [0 0 0 0 1 1]]
```

```
[5]: from sage.all import *
     n = m = 4
     a = genenerate_neighbord_matrix_m_n(m,n)
     print(a)
     A = Matrix(Integers(),a)
     Y = vector([1 for x in range(m*n)])
     Z = vector([0 for x in range(m*n)])
     X = A.solve_right(Y)
     print(X)
```

```
[[1 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0]
 [1 1 1 0 0 1 0 0 0 0 0 0 0 0 0 0]
 [0 1 1 1 0 0 1 0 0 0 0 0 0 0 0 0]
 [0 0 1 1 0 0 0 1 0 0 0 0 0 0 0 0]
 [1 0 0 0 1 1 0 0 1 0 0 0 0 0 0 0]
```

```
[0 1 0 0 1 1 1 0 0 1 0 0 0 0 0 0]
[0 0 1 0 0 1 1 1 0 0 1 0 0 0 0 0]
[0 0 0 1 0 0 1 1 0 0 0 1 0 0 0 0]
[0 0 0 0 1 0 0 0 1 1 0 0 1 0 0 0]
[0 0 0 0 0 1 0 0 1 1 1 0 0 1 0 0]
[0 0 0 0 0 0 1 0 0 1 1 1 0 0 1 0]
[0 0 0 0 0 0 0 1 0 0 1 1 0 0 0 1]
[0 0 0 0 0 0 0 0 1 0 0 0 1 1 0 0]
[0 0 0 0 0 0 0 0 0 1 0 0 1 1 1 0]
[0 0 0 0 0 0 0 0 0 0 1 0 0 1 1 1]
[0 0 0 0 0 0 0 0 0 0 0 1 0 0 1 1]]
(0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0)
```

5 Solution Amount

```
[6]: n = 9
     a = genenerate_neighbord_matrix(n)
     A = Matrix(Integers(2),a)
     print(2**A.kernel().dimension())
```

```
256
```

6 Benchmark

```
[7]: import datetime
     import numpy as np

     def matrix_solve(mat):
         A = Matrix(Integers(2),mat)
         Y = vector([1 for x in range(n**2)])
         Z = vector([0 for x in range(n**2)])
         X = A.solve_right(Y)
         return X

     val = []
     # run on range(10 ,61,5)
     for i,n in enumerate(range(10 ,15)):
         # print(i)
         mat = genenerate_neighbord_matrix(n)

         a0 = datetime.datetime.now()
         matrix_solve(mat)
         b0 = datetime.datetime.now()
         c0 = b0 - a0
         t0 = c0.total_seconds()
         # print(t0)
```

```
    a1 = datetime.datetime.now()
    generate_mat_spanish_alg(mat)
    b1 = datetime.datetime.now()
    c1 = b1 - a1
    t1 = c1.total_seconds()
    # print(t1)

    val.append((n, t0, t1))

res = np.array(val)
# np.savetxt("benchmark.csv", res, delimiter = ',')
print(res)
```

```
[[10.         0.020791  0.184697]
 [11.         0.029358  0.261447]
 [12.         0.0316    0.366729]
 [13.         0.045727  0.51665 ]
 [14.         0.068553  0.670478]]
```