Trabalho Prático 1

Bruno Jardim (a91680) José Ferreira (a91636)

2 - Sudoku

1. Da definição do jogo "Sudoku" generalizado para a dimensão ; o problema tradicional corresponde ao caso . O objetivo do Sudoku é preencher uma grelha de com inteiros positivos no intervalo até , satisfazendo as seguintes regras:
   - Cada inteiro no intervalo até ocorre só uma vez em cada coluna, linha e secção .
   - No início do jogo uma fração das casas da grelha são preenchidas de forma consistente com a regra anterior.

Objetivo do trabalho:

1. Construir um programa para inicializar a grelha a partir dos parâmetros e

1. Construir soluções do problema para as combinações de parâmetros e . Que conclusões pode tirar da complexidade computacional destas soluções.

Análise do Problema:

Existe um e um , onde corresponde ao tamanho do Sudoku que será representado e à percentagem de células com um valor predefinido.

Com estes valores iremos criar uma matriz que representa o Sudoku.

Para solucionar o problema iremos representar este Sudoku como sendo um grafo e utilizaremos uma versão alterada da função desenvolvida nas aulas, com a finalidade desta colorir o grafo com cores

Finalmente usaremos o grafo colorido para construir uma matriz correspondente ao Sudoku inicial resolvido.

Solução do Problema:

1. Construir um Sudoku a partir dos parâmetros e
   - Para isto usaremos as funções e , elas servem para criar uma matriz/Sudoku de tamanho com probabilidade de conter um número em cada célula e para verificar probabilidades, respetivamente
     - Não existe garantia que o Sudoku gerado tenha solução

```
def geraMat(tam,prob):
    size = pow(tam,2)
    mat = [[0 for i in range(size)] for j in range(size)]
    for y in range(len(mat)):
        for x in range(len(mat[y])):
            if probCheck(prob):
                num = random.randint(1, size)
                if(validaPos(mat,x,y,num)):
                    mat[y][x] = num
    return mat


def probCheck(prob):
n = random.random()
if(n <= prob):
    return True
return False
```

1. Construir o grafo equivalente à matriz do Sudoku
   - Para isto usaremos as funções e , que serve para criar as edges do grafo

```
def checkSquare(x1,y1,x2,y2,size):
        if x1//size == x2//size and y1//size == y2//size:
            return True
        return False




def createEdges(size):
    size = size**2
    mat = [x+1 for x in range(size**2)]
    mat = np.reshape(mat,(size,size))
    adj=[]
    for y in range(len(mat)):
        for x in range(len(mat[y])):
            for outros in range(x,len(mat[y])):
                adj.append((mat[y][x],mat[y][outros]))
    for y in range(len(mat)):
        for x in range(len(mat[x])):
            for outros in range(y,len(mat)):
                adj.append((mat[y][x],mat[outros][x]))

    for y in range(len(mat)):
        for x in range(len(mat[y])):
            for yy in range(len(mat)):
                for xx in range(len(mat[yy])):
                    if checkSquare(x,y,xx,yy,int(math.sqrt(len(mat)))) and (mat[y][x],mat[yy][xx]) not in adj:
                        adj.append((mat[y][x],mat[yy][xx]))

    edges = [(x,y) for (x,y) in adj if x!=y and (y,x)]
    return edges
```

1. Criar o grafo a partir das edges
   - Para isto usaremos a biblioteca

```
grafo=nx.Graph(edges)
```

1. Colorir o grafo
   - Para isto usaremos a função desenvolvida nas aulas, com algumas alterações. E a função para passar a informação da matriz para o grafo

```
def conv_info(mat, graph):
    for row in range( len(mat) ):
        for col in range( len(mat[row]) ):
            id_node = row*len(mat) + col + 1
            graph.nodes[id_node]['color'] = mat[row][col]

def ip_color(graph,k):


    # criar solver
    solver = pywraplp.Solver('BOP', pywraplp.Solver.BOP_INTEGER_PROGRAMMING)
    #criar dicionario de variaveis x{i,j}
    x = {}
    for i in graph:
        x[i] = {}
        for j in range(k):
            x[i][j] = solver.IntVar(0,1,'x[%i][%i]' % (i,j))

    # vertices pre-preenchidos
    for o in graph:
        if graph.nodes[o]['color'] != 0:
            c = graph.nodes[o]['color']
            solver.Add(x[o][c-1] == 1)




    # vertices adjacentes tem cores diferentes
    for o in graph:
        for d in graph[o]:
            for j in range(k):
                solver.Add(x[o][j] + x[d][j] <= 1)




    # cada vertice adjacente tem cores diferentes

    for i in graph:
        solver.Add(sum([x[i][j] for j in range(k)]) == 1)  # ou solver.Add(sum(list(x[i].values())) == 1)

    # invocar solver e colorir o grafo

    stat = solver.Solve()
    if stat == pywraplp.Solver.OPTIMAL:

      # colorir

        for i in graph:
            for j in range(k):
                if round(x[i][j].solution_value()) == 1:
                    graph.nodes[i]['color'] = j+1
        return True
    else:
        return False
```

1. Converter o grafo a matriz
   - Para isto usaremos a função , que recebe um grafo colorido e converte para uma matriz equivalente ao sudoku resolvido

```
def converte(grafo, size):
    cores=[]
    for i in grafo:
        cores.append(grafo.nodes[i]['color'])
    solvedSudoku=np.reshape(cores,(int(math.sqrt(len(cores))),int(math.sqrt(len(cores)))))
    return solvedSudoku
```

Funcões

In [2]:
```
!pip install ortools
```

```
Requirement already satisfied: ortools in c:\users\bruno\anaconda3\lib\site-packages (9.1.9490)
Requirement already satisfied: protobuf>=3.18.0 in c:\users\bruno\anaconda3\lib\site-packages (from ortools) (3.18.1)
Requirement already satisfied: absl-py>=0.13 in c:\users\bruno\anaconda3\lib\site-packages (from ortools) (0.14.1)
Requirement already satisfied: six in c:\users\bruno\anaconda3\lib\site-packages (from absl-py>=0.13->ortools) (1.15.0)
```

In [3]:
```
import numpy as np
import math
import networkx as nx
import time
from random import randint
import random
from ortools.linear_solver import pywraplp
```

In [4]:
```
def checkSquare(x1,y1,x2,y2,size):
        if x1//size == x2//size and y1//size == y2//size:
            return True
        return False

def createEdges(size):
    size = size**2
    mat = [x+1 for x in range(size**2)]
    mat = np.reshape(mat,(size,size))
    adj=[]
    for y in range(len(mat)):
        for x in range(len(mat[y])):
            for outros in range(x,len(mat[y])):
                adj.append((mat[y][x],mat[y][outros]))
    for y in range(len(mat)):
        for x in range(len(mat[x])):
            for outros in range(y,len(mat)):
                adj.append((mat[y][x],mat[outros][x]))

    for y in range(len(mat)):
        for x in range(len(mat[y])):
            for yy in range(len(mat)):
                for xx in range(len(mat[yy])):
                    if checkSquare(x,y,xx,yy,int(math.sqrt(len(mat)))) and (mat[y][x],mat[yy][xx]) not in adj:
                        adj.append((mat[y][x],mat[yy][xx]))

    edges = [(x,y) for (x,y) in adj if x!=y and (y,x)]
    return edges
```

In [5]:
```
def ip_color(graph,k):


    # criar solver
    solver = pywraplp.Solver('BOP', pywraplp.Solver.BOP_INTEGER_PROGRAMMING)
    #criar dicionario de variaveis x{i,j}
    x = {}
    for i in graph:
        x[i] = {}
        for j in range(k):
            x[i][j] = solver.IntVar(0,1,'x[%i][%i]' % (i,j))

    # vertices pre-preenchidos
    for o in graph:
        if graph.nodes[o]['color'] != 0:
            c = graph.nodes[o]['color']
            solver.Add(x[o][c-1] == 1)




    # vertices adjacentes tem cores diferentes
    for o in graph:
        for d in graph[o]:
            for j in range(k):
                solver.Add(x[o][j] + x[d][j] <= 1)




    # cada vertice adjacente tem cores diferentes

    for i in graph:
        solver.Add(sum([x[i][j] for j in range(k)]) == 1)  # ou solver.Add(sum(list(x[i].values())) == 1)

    # invocar solver e colorir o grafo

    stat = solver.Solve()
    if stat == pywraplp.Solver.OPTIMAL:

      # colorir

        for i in graph:
            for j in range(k):
                if round(x[i][j].solution_value()) == 1:
                    graph.nodes[i]['color'] = j+1
        return True
    else:
        return False
```

In [6]:
```
def conv_info(mat, graph):
    for row in range( len(mat) ):
        for col in range( len(mat[row]) ):
            id_node = row*len(mat) + col + 1
            graph.nodes[id_node]['color'] = mat[row][col]
```

In [7]:
```
def converte(grafo, size):
    cores=[]
    for i in grafo:
        cores.append(grafo.nodes[i]['color'])
    solvedSudoku=np.reshape(cores,(int(math.sqrt(len(cores))),int(math.sqrt(len(cores)))))
    return solvedSudoku
```

In [8]:
```
def probCheck(prob):
    n = random.random()
    if(n <= prob):
        return True
    return False
```

In [9]:
```
def validaPos(mat,x,y,n):
    tam = len(mat)
    for xver in range(tam):
        if mat[y][xver] == n:
            return False
    for yver in range(tam):
        if mat[yver][x] == n:
            return False
    sqSize = int(math.sqrt(tam))
    # Check Square
    inicioLin = y - y % sqSize
    inicioCol = x - x % sqSize
    for i in range(sqSize):
        for j in range(sqSize):
            if mat[i + inicioLin][j + inicioCol] == n:
                return False
    return True
def geraMat(tam,prob):
    size = pow(tam,2)
    mat = [[0 for i in range(size)] for j in range(size)]
    for y in range(len(mat)):
        for x in range(len(mat[y])):
            if probCheck(prob):
                num = random.randint(1, size)
                if(validaPos(mat,x,y,num)):
                    mat[y][x] = num
    return mat
```

In [10]:
```
def checkZero(mat):
    for y in range( len(mat) ):
        for x in range( len(mat[y]) ):
            if mat[y][x]==0:
                return False
    return True
```

In [11]:
```
def display(mat):
    print('\n'.join([''.join(['{:4}'.format(item) for item in row]) for row in mat]))
def sol():
    print()
    print('SOLUÇÃO:')
    print()
```

Execução do código para n = 3.

In [12]:
```
edges=createEdges(3)
grafo5=nx.Graph(edges)
mat = geraMat(3, 0.2)

for x in mat:
    print(x)
print('\n')

conv_info(mat, grafo5)
ti = time.perf_counter()
ip_color(grafo5,9)
tf = time.perf_counter()
solved = converte(grafo5, 9)
if checkZero(solved):
    display(solved)

else: print("Não há solução.")


print(f"Time {tf - ti: 0.5f} seconds")
```

```
[0, 4, 0, 0, 0, 0, 0, 2, 0]
[0, 8, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 5, 0, 0, 0, 0, 1]
[7, 0, 2, 0, 0, 1, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0]
[4, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 5, 3, 0, 0, 2, 0, 0, 0]
[8, 0, 0, 0, 0, 0, 0, 3, 0]


   6   4   5   1   9   7   3   2   8
   3   8   1   2   4   6   5   9   7
   2   7   9   5   3   8   6   4   1
   7   3   2   8   6   1   4   5   9
   5   1   4   3   2   9   8   7   6
   9   6   8   4   7   5   2   1   3
   4   2   6   7   1   3   9   8   5
   1   5   3   9   8   2   7   6   4
   8   9   7   6   5   4   1   3   2
Time  0.18866 seconds
```

Execução do código para n = 4.

In [24]:
```
edges=createEdges(4)
grafo5=nx.Graph(edges)
mat = geraMat(4, 0.6)

display(mat)
sol()
conv_info(mat, grafo5)
ti = time.perf_counter()
ip_color(grafo5,16)
tf = time.perf_counter()
display(converte(grafo5, 16))
print(f"Time {tf - ti: 0.5f} seconds")
```

```
   0   0   0   2   0   6   7  15   0   0   0   0   0   0  10  16
   6  10   0   0   1   2   4   0  15  13   0  11   0   0   0   0
   0   0  12   0  14   0   0   0   3   0   0   6   1   0   0  11
   0  15   0   1   0  16   0   0   0   0   0   0  12   0   0   8
  11   0   9   0   5   0   0   6   7  14  13   0   8   0   3   0
  10   0   0  15   0   0  16   0   0   0   4   0   2   0   0  12
   0   2  16   4   8  15   1   0   0   0  12   0   0   0   0   0
   3   0   0   7   0   0   0   0   0   1   6   0   0   0  11   0
   0   0  15   0  13   8   0  10   2   0   0  16   0   0   0   9
   0   9   8   0  12   5   0   0   0   0   0   7   0   0   0   0
   0   0   6  16  15   0   0   0   9  12   0   4   0   0   0  14
   0   0   0  14   0   0   0   0  10   0   0   3   0   0   0   7
   0   0   7   0   0   1   0   3   0  11   0   0   0   0  14   0
   9   0   0   0   0  14   0   0  12   6   1   0   0   0   4   0
   0   0   5   3   0   0   0   0   0   0   0   9   0  13  15  10
   2   8   0   0  11  10   9   0   0   0   0  15   0   0  16   0

SOLUÇÃO:

   0   0   0   2   0   6   7  15   0   0   0   0   0   0  10  16
   6  10   0   0   1   2   4   0  15  13   0  11   0   0   0   0
   0   0  12   0  14   0   0   0   3   0   0   6   1   0   0  11
   0  15   0   1   0  16   0   0   0   0   0   0  12   0   0   8
  11   0   9   0   5   0   0   6   7  14  13   0   8   0   3   0
  10   0   0  15   0   0  16   0   0   0   4   0   2   0   0  12
   0   2  16   4   8  15   1   0   0   0  12   0   0   0   0   0
   3   0   0   7   0   0   0   0   0   1   6   0   0   0  11   0
   0   0  15   0  13   8   0  10   2   0   0  16   0   0   0   9
   0   9   8   0  12   5   0   0   0   0   0   7   0   0   0   0
   0   0   6  16  15   0   0   0   9  12   0   4   0   0   0  14
   0   0   0  14   0   0   0   0  10   0   0   3   0   0   0   7
   0   0   7   0   0   1   0   3   0  11   0   0   0   0  14   0
   9   0   0   0   0  14   0   0  12   6   1   0   0   0   4   0
   0   0   5   3   0   0   0   0   0   0   0   9   0  13  15  10
   2   8   0   0  11  10   9   0   0   0   0  15   0   0  16   0
Time  2.15219 seconds
```

Execução do código para n = 5.

In [28]:
```
edges=createEdges(5)
grafo5=nx.Graph(edges)
mat = geraMat(5, 0.6)

display(mat)
sol()
conv_info(mat, grafo5)
ti = time.perf_counter()
ip_color(grafo5,25)
tf = time.perf_counter()
display(converte(grafo5, 25))
print(f"Time {tf - ti: 0.5f} seconds")
```

```
   0  25   9   0   0  24   6  11  22   0   0  23   0   0   0   0   0   0   0   0  20   0   0   3   0
   2   0  22   1  20   0   7   0  16   0  13   0   0  18   0   0   0  14   8   0   0  10   0   4   0
  16   0   0   0   0   0   0  18   8   0   6   0  20   0   1  11   0  10   0   3   0   0   0   0   0
  19   0   0   0   0   0  23  14   0   5   0   0   0   0   0   0   0   0  15  22  17  25   6   0   0
   8  10   0   4   0   0   0   0   0   0   5   0  11   9   0   0   0   0   0   0   0  19   1   0  16
   0   3   0  16  11   0   0   9   7  14   0   0   0   0  17   1   6   0   0  25   0  22   0  10   0
  20   8  10   0   4   5   0  22  15  16   2   7   6   1  14  21   0  19   0  12   0   0   0   0   0
   0   0   0   0   0   4   0  12   0   0   0   0   0  19   0   0   0  18  22  16   0  14   0   6   0
   0  21   0  18  14   3   0  13   0   0  10   0   0   0   0   0   0  24  11   0   5  16   0   0  20
  13   0  17   0   2   0  24   0  20   0   0   0  21  15   0   0   0   5   3   0  23   0   0   0   0
  23   0   0  22  15   0  17   0   0   0   0  25   0   0   9  20   2   0   0  19   1   0   0   0  13
   0   0   0   0   0  25   0   0   0   0   8   1   0   0   0   0   0  13  21   0  24   0   0   0  18
   0   0   0   0  13   0   0  23   0   0   4   0   0   0  12   0  18  22   0   0   0  15  21   0   0
  12   5  18   7   9   0   0   0   0  24   0   0   0  22   0  25   0   0   4   0   0   0   0   0   8
   0   0   3   8   1   0   0   0   0  19  15   2   0   0   0   0   0  16   0   0   0  17   9  11   0
   0   0   0  13   0  19   0  10  25   0   0  15   5  14  24  12   0   0  23   0   0   0   0   0   1
   0   0  14  17   0   0   0   0   0   0   0   0   3  25   0   0   0   4  13  21   0   0   7   0   0
   0   0   0   0   0   0   0  21   2   0   0   0   0   0   0   0   0   0   0   0   0   9  15  18   0
   6  23   0   0  24   0   0   0   0  19   0   2  10  22   0   0   0   0   0   0   0   0   3   0   5
   0  11   0   0  21   0   0   5   0  13   0   0   9   0   0  16  20   3   0   0   6   0   0   8   0
   0  16   8  14   0   0  12   0   0   0   0   0   0  21   4   0   7   9   0  20   0  11   0   0  10
   0   0  23  21   0  15   0   0  17   2  24   0   0   0   0   0   0   0   0   0   0   4  16   0   0
  25   4   2   0  18   0   0   0  11   0   0   9   0   0   0  10   0   0   0   0   0   0   0   0  24
   0   0   0   0   0   0   0   0  18  23   0   0   0   3   0   0   0   0   0   0   0   0  14   7   0
   0   0   0  24  17   0   0   0   0  22   0  11   0   5   0  18   0   0   0   0   0   0  20  12   0

SOLUÇÃO:

  15  25   9   5   7  24   6  11  22  21  14  23   4   2  10  17  19   1  16  13  20   8  18   3  12
   2  17  22   1  20  12   7  19  16  15  13  21  25  18   3   6  24  14   8   5  11  10  23   4   9
  16  14  24  12  23   2   9  18   8   4   6  17  20  19   1  11  21  10  25   3   7   5  13  15  22
  19  18  13  11   3   1  23  14  10   5   7  12   8  24  16   4   9  20  15  22  17  25   6  21   2
   8  10  21   4   6  20  13  17   3  25   5  22  11   9  15  23  12   7   2  18  14  19   1  24  16
  24   3  19  16  11  18  21   9   7  14  23   5  12  13  17   1   6   2  20  25   8  22   4  10  15
  20   8  10  23   4   5  25  22  15  16   2   7   6   1  14  21  17  19   9  12   3  18  24  13  11
   1   7   5  15  25   4   2  12  23   8  20   3  24  11  19  13  10  18  22  16   9  14  17   6  21
   9  21  12  18  14   3  19  13   6  17  10   4  22   8  25   7  23  24  11  15   5  16   2   1  20
  13  22  17   6   2  11  24   1  20  10   9  16  21  15  18   8   4   5   3  14  23  12  19  25   7
  23  24   4  22  15  10  17  16  21  12  11  25  18   6   9  20   2   8   7  19   1   3   5  14  13
  17   2  11  10  16  25  15   3   4  20   8   1  19  23   5   9  14  13  21   6  24   7  12  22  18
  14  19  25  20  13   8  11  23   5   9   4  24   7  17  12   3  18  22   1  10  16  15  21   2   6
  12   5  18   7   9   6   1   2  13  24   3  14  16  22  21  25  15  17   4  11  19  23  10  20   8
  21   6   3   8   1  22  18   7  14  19  15   2  10  20  13  24   5  16  12  23  25  17   9  11   4
  18   9  16  13   8  19  20  10  25   3  21  15   5  14  24  12  22   6  23   7   4   2  11  17   1
  22  12  14  17   5   9  16  15  24  18   1   8   3  25   6   2  11   4  13  21  10  20   7  19  23
   4  20   1   3  19   7  22  21   2   6  17  13  23  16  11   5   8  25  10  24  12   9  15  18  14
   6  23   7  25  24  17   8   4  12  11  19  20   2  10  22  14   1  15  18   9  13  21   3  16   5
  10  11  15   2  21  23  14   5   1  13  12  18   9   4   7  16  20   3  19  17   6  24  22   8  25
   3  16   8  14  22  13  12  24  19   1  18   6  17  21   4  15   7   9   5  20   2  11  25  23  10
  11   1  23  21  10  15   5  25  17   2  24  19  14   7  20  22  13  12   6   8  18   4  16   9   3
  25   4   2  19  18  14   3  20  11   7  22   9  13  12  23  10  16  21  17   1  15   6   8   5  24
   5  13  20   9  12  21   4   6  18  23  16  10  15   3   8  19  25  11  24   2  22   1  14   7  17
   7  15   6  24  17  16  10   8   9  22  25  11   1   5   2  18   3  23  14   4  21  13  20  12  19
Time  16.92033 seconds
```

Execução do código para n = 6.

In [ ]:
```
edges=createEdges(6)
grafo5=nx.Graph(edges)
mat = geraMat(6, 0)

for x in mat:
    print(x)
print('\n')

conv_info(mat, grafo5)
ti = time.perf_counter()
ip_color(grafo5,36)
tf = time.perf_counter()
print(converte(grafo5, 36))
print(f"Time {tf - ti: 0.5f} seconds")
```

```
[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, 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, 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, 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, 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, 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, 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, 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, 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]
[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, 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, 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, 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, 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, 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, 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, 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, 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]
[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, 0, 0, 0, 0]
```

Versão Backtracking

- Definimos também uma função de resolução de Sudokus com um algoritmo com uma estratégia de Backtracking.
  - Apesar de ser mais eficiente em , para valores de o tempo de execução não é viável

In [54]:
```
def valido(mat,x,y,n):
    tam = len(mat)
    for xver in range(tam):
        if mat[y][xver] == n:
            return False
    for yver in range(tam):
        if mat[yver][x] == n:
            return False



    sqSize=int(math.sqrt(tam))
    #Check Square
    inicioLin=y - y % sqSize
    inicioCol=x - x % sqSize



    for i in range(sqSize):
        for j in  range(sqSize):
            if mat[i+inicioLin][j+inicioCol] == n:
                return False
    return True



def sudokuSolver(mat, x, y):
    fim = len(mat)
    #Verificamos se chegamos ao final da matriz, se sim encontramos uma solucao

    if y == fim-1 and x == fim:
        return True
    #Se chegarmos ao final da linha voltamos ao inicio na linha seguinte
    if x == fim:
        y+=1
        x=0
    #Se o numero estiver preenchido saltamos a frente pois é um dos pre-preenchidos
    if mat[y][x] > 0 :
        return sudokuSolver(mat,x+1,y)
    #Tentamos arranjar solucao para o sudoku
    for n in range(1,fim+1):
        #Verificamos se o numero que tentamos preencher e valido
        if valido(mat,x,y,n):
            #Preenchemos com o numero caso tenha sido validado
            mat[y][x] = n
            #Verificamos se e possivel prosseguir, caso contrario damos backtrack
            if sudokuSolver(mat,x+1,y):
                return True
        mat[y][x] = 0
    return False
```

In [81]:
```
mat=geraMat(3,0.2)
display(mat)
sol()
ti = time.perf_counter()
sudokuSolver(mat,0,0)
tf = time.perf_counter()
display(mat)
print(f"Time {tf - ti: 0.5f} seconds")
```

```
   0   6   0   0   0   0   0   0   9
   0   0   1   0   9   0   0   7   3
   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0
   3   0   0   2   0   0   0   0   0
   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   1   7
   0   0   0   8   0   3   0   0   0
   0   0   0   0   0   0   0   2   0

SOLUÇÃO:

   2   6   3   1   4   7   5   8   9
   4   5   1   6   9   8   2   7   3
   7   8   9   3   2   5   1   4   6
   1   2   4   5   3   6   7   9   8
   3   7   6   2   8   9   4   5   1
   8   9   5   7   1   4   6   3   2
   6   4   8   9   5   2   3   1   7
   5   1   2   8   7   3   9   6   4
   9   3   7   4   6   1   8   2   5
Time  0.01867 seconds
```

Gráficos

1. Comparação dos diferentes solvers para alt text

alt text

1. Comparação do solver para alt text

In [ ]: