In [1]:

```
labirinto = [
  ['x','x','x','x','x','x','x','x','x','x','x','x','x','x','x','x','x','x'],
  ['x','A',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ','x'],
  ['x',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ','x'],
  ['x',' ',' ',' ','x','x','x','x','x',' ',' ',' ',' ',' ',' ',' ',' ','x'],
  ['x',' ',' ',' ',' ',' ','0',' ',' ',' ','x','x','x','x','x',' ',' ','x'],
  ['x',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ','x',' ',' ',' ',' ',' ','x'],
  ['x',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ','x',' ',' ','0',' ',' ','x'],
  ['x','x','x','x','x','x','x','x','x','x','x','x','x','x','x','x','x','x']
]
class State:
    def __init__(self, pai, linha, coluna, objetivos):
        self.pai = pai
        self.linha = linha
        self.coluna = coluna
        self.objetivos = objetivos

def showState(s):
    #print("(", s.linha, ",", s.coluna, ")")
    labirinto[s.linha][s.coluna] = '#'

def initialState():
    return State(None, 1, 1, [])

def goal(s):
    #return labirinto[s.linha][s.coluna] == 'G'
    return len(s.objetivos) == 2

def move(s, mLinha, mColuna):
    linhaO = s.linha + mLinha
    colunaO = s.coluna + mColuna

    objetivos = s.objetivos.copy()
    if(labirinto[linhaO][colunaO] == 'G'):
        if(not((linhaO, colunaO) in objetivos)):
            objetivos.append((linhaO, colunaO))

    if(labirinto[linhaO][colunaO] == 'x'):
        return None

    return State(s, linhaO, colunaO, objetivos)

def cima(s):
    objetivos = s.objetivos.copy()
    if(labirinto[s.linha - 1][s.coluna] == 'G'):
        if(not((s.linha - 1, s.coluna) in objetivos)):
            objetivos.append((s.linha - 1, s.coluna))
    if(labirinto[s.linha - 1][s.coluna] == 'x'):
        return None
    return State(s, s.linha - 1, s.coluna, objetivos)

def baixo(s):
    objetivos = s.objetivos.copy()
    if(labirinto[s.linha + 1][s.coluna] == 'G'):
        if(not((s.linha + 1, s.coluna) in objetivos)):
            objetivos.append((s.linha + 1, s.coluna))

    if(labirinto[s.linha + 1][s.coluna] == 'x'):
        return None
    return State(s, s.linha + 1, s.coluna, objetivos)

def direita(s):
    objetivos = s.objetivos.copy()
```

```
    if(labirinto[s.linha][s.coluna + 1] == 'G'):
        if(not((s.linha, s.coluna + 1) in objetivos)):
            objetivos.append((s.linha, s.coluna + 1))

    if(labirinto[s.linha][s.coluna + 1] == 'x'):
        return None
    return State(s, s.linha, s.coluna + 1, objetivos)

def esquerda(s):
    objetivos = s.objetivos.copy()
    if(labirinto[s.linha][s.coluna - 1] == 'G'):
        if(not((s.linha, s.coluna - 1) in objetivos)):
            objetivos.append((s.linha, s.coluna - 1))

    if(labirinto[s.linha][s.coluna - 1] == 'x'):
        return None
    return State(s, s.linha, s.coluna - 1, objetivos)

def expand(s):
    ret = []
    filho = move(s, -1, 0)
    if(filho != None):
        ret.append(filho)
    filho = move(s, 1, 0)
    if(filho != None):
        ret.append(filho)
    filho = move(s, 0, 1)
    if(filho != None):
        ret.append(filho)
    filho = move(s, 0, -1)
    if(filho != None):
        ret.append(filho)
    return ret

def showPath(s):
    if(s == None):
        return
    showPath(s.pai)
    showState(s)

def igual(s1, s2):
    return (s1.linha == s2.linha and s1.coluna == s2.coluna and set(s1.objetivos) == set

def ancestral(pai, filho):
    if(pai == None):
        return False

    if(igual(pai, filho)):
        return True
    return ancestral(pai.pai, filho)

queue = []

def enqueue(s):
    queue.append(s)

def dequeue():
    return queue.pop(0)

visitados = 0
s = initialState()

def naFronteira(s):
    for aux in queue:
```

```
        if(igual(s, aux)):
            return True
    return False

expandidos = []

def jaExpandiu(s):
    for aux in expandidos:
        if(igual(s, aux)):
            return True
    return False


enqueue(s)
while(queue):
    s = dequeue()

    if(goal(s)):
        showPath(s)
        break

    filhos = expand(s)
    expandidos.append(s)
    visitados += 1
    #print(s.contador)
    for filho in filhos:
        if(not (ancestral(s.pai, filho) or naFronteira(filho) or jaExpandiu(filho))):
            enqueue(filho)

print("Quantidade de estados expandidos: ", visitados)
for linha in labirinto:
    print(linha)
```

```
Quantidade de estados expandidos:  84
['x', 'x', 'x', 'x', 'x', 'x', 'x', 'x', 'x', 'x', 'x', 'x', 'x', 'x', 'x', 'x',
'x']
['x', 'A', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ',
'x']
['x', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ',
'x']
['x', ' ', ' ', ' ', 'x', 'x', 'x', 'x', 'x', ' ', ' ', ' ', ' ', ' ', ' ', ' ',
'x']
['x', ' ', ' ', ' ', ' ', ' ', '0', ' ', ' ', ' ', 'x', 'x', 'x', 'x', 'x', ' ', ' ',
'x']
['x', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', 'x', ' ', ' ', ' ', ' ', ' ', ' ',
'x']
['x', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', 'x', ' ', ' ', '0', ' ', ' ', ' ',
'x']
['x', 'x', 'x', 'x', 'x', 'x', 'x', 'x', 'x', 'x', 'x', 'x', 'x', 'x', 'x', 'x', 'x',
'x']
```