$Logica\ Computacional\ 02 - Outubro - 2023\ LCC$

**Grupo 06:**


- João Manuel Franqueira da Silva, A91638
- Eduardo Manuel Sousa Pereira, A70619


# $TP1 - Problema\ 2$

Um sistema de tráfego é representado por um grafo orientado ligado. Os nodos denotam pontos de acesso e os arcos denotam vias de comunicação só com um sentido . O grafo tem de ser ligado: entre cada par de nodos *<n1,n2>* tem de existir um caminho $n_1 \rightsquigarrow n_2$ e um caminho $n_2 \rightsquigarrow n_1$.

In [89]:
```python
from ortools.linear_solver import pywraplp
import random
from pysmt.shortcuts import Symbol, LE, GE, Int, And, Equals, Plus, Solver, Not, Or, (
from pysmt.typing import INT
import networkx as nx
import matplotlib.pyplot as plt
import copy
```

Gerar aleatoriamente o grafo com $N \in 8, \ldots, 15$ nodos e com ramos verificando:

1. Cada nodo tem um número aleatório de descendentes $d \in \{1, \ldots, 3\}$ cujos destinos são também gerados aleatoriamente.
2. Se existirem "loops" ou destinos repetidos, deve-se gerar outro grafo.

O seguinte problema, pode ser traduzido então numa matriz quadrada $\mathrm{NxN}$ preenchida por zeros e uns, em que $\mathrm{matriz[i][j]} == 1$ denota a existência de uma aresta do nodo i para o nodo j. Para evitar a existência de loops no grafo, temos que prevenir que $\mathrm{matriz[i][i]} == 1$ para qualquer $\mathrm{i}$.

O problema pode ser descrito por:

$$0 < \forall_{i=0}^{n-1} \sum_{j=0}^{n-1} matriz[i][j] \leq 3$$

$$\forall_{i=0}^{n-1} matriz[i][i] \neq 0$$

De seguida, traduzimos a nossa matriz para um grafo, adicionando os nodos e vértices de $\mathrm{i}$ para $\mathrm{j}$ quando $\mathrm{matriz[i][j]} = 1$ . Por último, verificamos se o grafo é conexo, chamando a função $\mathrm{nx.is\_strongly\_connected()}$. Se o grafo não for conexo, voltamos a formar outra matriz, até encontrarmos uma que represente um grafo conexo.

In [90]:
```python
nodos=random.randint(8,15)

while True:
    nodos=5
```

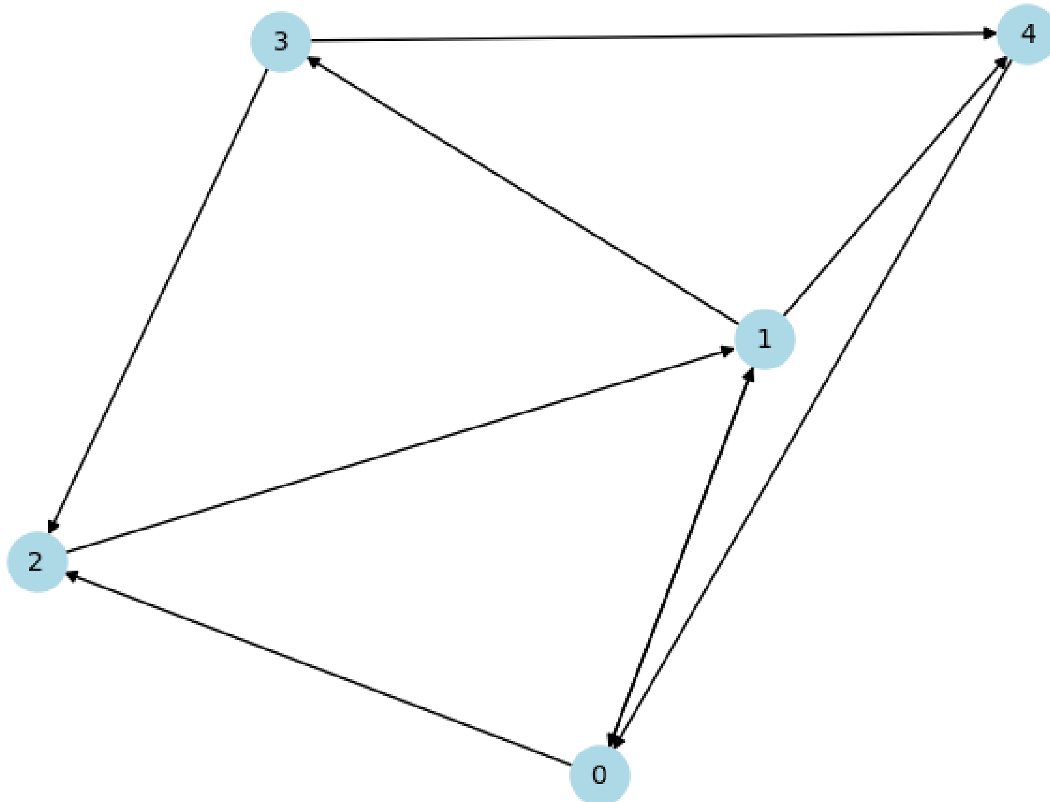
```python
    grafo=nx.DiGraph()

    for n in range(nodos):
        grafo.add_node(n)

    matriz = []
    for i in range(nodos):
        row = [0] * nodos
        matriz.append(row)

    for v1 in range(nodos):
        x=0
        arco=random.randint(1,3)

        while x<arco:
            v2=random.randint(0,nodos-1)
            if v2!=v1 and matriz[v1][v2]==0:
                matriz[v1][v2]=1
                x=x+1

    for v1 in range(nodos):
        for v2 in range(nodos):
            if v1!=v2:
                if matriz[v1][v2]==1:
                    grafo.add_edge(v1,v2)
    r=nx.is_strongly_connected(grafo)


    if r==True:
        break
```

# Desenho do grafo gerado e informação sobre os nodo

In [91]:

```python
print(nx.to_dict_of_lists(grafo))
nx.draw(grafo, with_labels=True, node_size=500, node_color='lightblue', font_size=10)
plt.show()
```

```
{0: [1, 2], 1: [0, 3, 4], 2: [1], 3: [2, 4], 4: [0]}
```




Pretende-se fazer manutenção interrompendo determinadas vias.
Determinar o maior número de vias que é possível remover mantendo o grafo ligado:

A seguinte função, calcula o maior número de arestas que se podem remover enquanto o grafo continua conexo, fazendo uma cópia do grafo e retirando arestas.

Se esta cópia continuar conexa, entao é feita uma chamada recursiva e o valor de $x$ incrementado, quando a remoção de uma aresta resulta num grafo nao conexo, não é feita nenhuma chamada recursiva, e o valor de $x$ descartado.

O dicionário $memo$, é utilizado para não realizar computações repetidas sobre o resultado do atual valor de x, tornando a função mais eficiente.

Por último, o valor de $y$ (número de arestas já removidas) é comparado com o $valormax$ (maior número de arestas já retiradas) e, caso seja maior, $valormax$ é atualizado.

In [92]:
```
#esta funçao calcula o maior numero de arestas que se podem remover, enquanto o grafo
def func(grafo, x, memo={}):
    if x in memo:
        return memo[x]

    valormax = x
    for (o, d) in grafo.edges():
        gr = grafo.copy()
        gr.remove_edge(o, d)
        if nx.is_strongly_connected(gr):
```

```
            y = func(gr, x + 1, memo)
            if y > valormax:
                valormax = y


    memo[x] = valormax
    return valormax
```

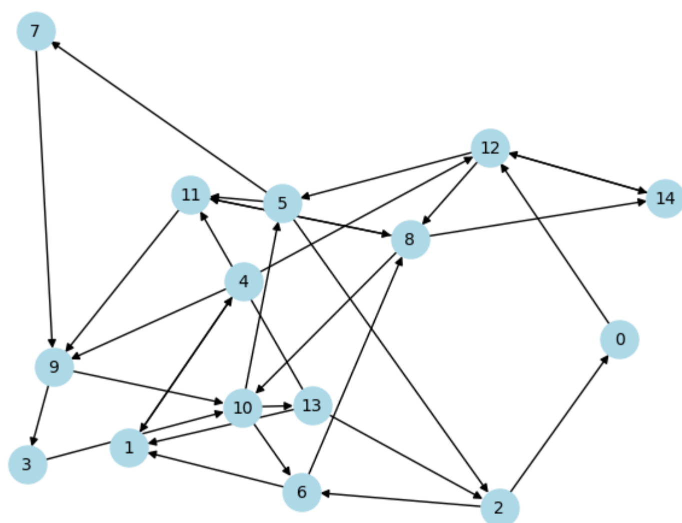
Número de arestas que é possivel retirar:

In [93]:
```
g2 = grafo.copy()
r=func(g2, 0)
print("Maior número de arestas retiradas ao grafo de forma a continuar conexo: %s"%(r)
```

```
Maior número de arestas retiradas ao grafo de forma a continuar conexo: 4
```

# Exemplo 1

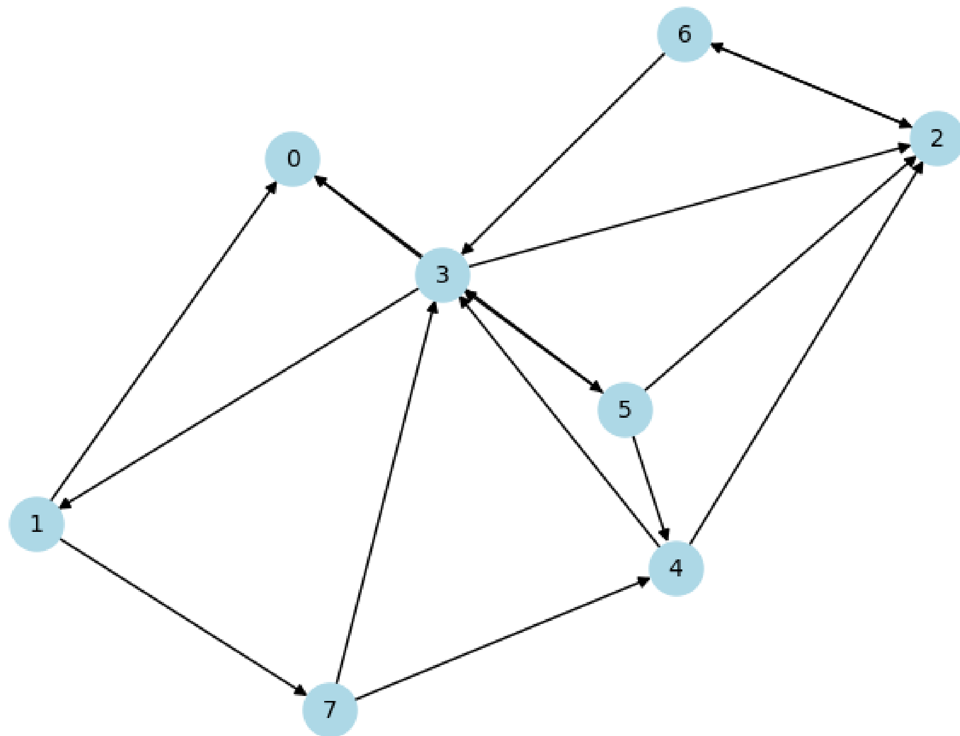
```
{0: [12], 1: [4], 2: [0, 6], 3: [10], 4: [1, 9, 12], 5: [2, 7, 11], 6: [1, 8], 7: [9], 8: [10, 11, 14], 9: [3, 10], 10: [5, 6, 13], 11: [8, 9], 12: [5, 8, 14], 13: [1, 2, 11], 14: [12]}
```




```
Maior numero possivel de arestas retiradas enquanto o grafo continua conexo: 13
```

# Exemplo 2

```
{0: [5], 1: [0, 7], 2: [6], 3: [0, 1, 2], 4: [2, 3], 5: [2, 3, 4], 6: [2, 3], 7: [3, 4]}
```




```
Maior numero possivel de arestas retiradas enquanto o grafo continua conexo: 7
```