Case Vallourec

Tasso Augusto Tomaz Pimenta 2021072198

Data entregue para o case:

```
import pandas as pd

df = pd.DataFrame({
    "Atividade": ["A", "B", "C", "D", "E", "F", "G", "H", "I", "J", "K", "L", "M", "N"],
    "Descrição": [  "Produção de Aço"           , #A
                    "Laminação do Tubo"         , #B
                    "Tratamento termico"        , #C
                    "Laboratório de superfícies", #D
                    "Transporte entre planta"   , #E
                    "Usinagem"                  , #F
                    "Rosqueadeira"              , #G
                    "Preparação para entrega"   , #H
                    "Laboratório de amostra"    , #I
                    "Laminação da Luva"         , #J
                    "Tratamento térmico da Luva", #K
                    "Laboratorio da luva"       , #L
                    "Acabamento da Luva"        , #M
                    "Acoplamento"]              , #N
    "Atividades Predecessoras": [   " ",        #A
                                    "A",        #B
                                    "B",        #C
                                    "C",        #D
                                    "C",        #E
                                    "E",        #F
                                    "F",        #G
                                    "D,I,L,N",  #H
                                    "F",        #I
                                    "A",        #J
```

```
                                    "J",        #K
                                    "K",        #L
                                    "K",        #M
                                    "G,M"       #N
                                    ],
    "Duração [Dias]": [25, 21, 23, 32, 13, 2, 2, 2, 32, 7, 8, 32, 2, 3]
})
```

Modelo usando o pulp

```
def caminho_critico_pulp(df):
    import pulp

    atividades      = df['Atividade'].tolist()
    duracoes        = df['Duração [Dias]'].tolist()
    predecessores   = df['Atividades Predecessoras'].tolist()

    problem = pulp.LpProblem("Caminho Crítico", pulp.LpMinimize)
    x       = pulp.LpVariable.dicts("x", atividades, lowBound=0, cat=pulp.LpInteger)

    for i, atividade in enumerate(atividades):
        for predecessor in predecessores[i].split(','):
            if predecessor != ' ':
                problem += x[atividade] >= x[predecessor] + duracoes[atividades.index(predec

    problem += pulp.lpSum([x[atividade] for atividade in atividades])
    problem.solve()

    caminho_critico = {}
    for atividade in atividades:
            caminho_critico[atividade] = x[atividade].varValue
    return caminho_critico
```

Só para explicar a formulação: basicamente quero minimizar o somatorio dos x, ou seja quero pegar o caminho minimo para realização de cada atividade as restrições são: tempo para realizar atividade tem que ser maior ou igual ao tempo para relizar a atividade predecessora de maior valor + a duração do predecessor . x representa o valor da realização da atividade predecessora maior, então para recuperar o tempo da ativiade eu somo com o tempo de realizar ela mesmo # modelo usando grafos(networkx)

```python
def grafo_caminho_critico(df):
    import networkx as nx
    import copy
    atividades      = df['Atividade'].tolist()
    #duracoes        = df['Duração [Dias]'].tolist()
    predecessores   = df['Atividades Predecessoras'].tolist()

    grafo = nx.DiGraph()
    for i, atividade in enumerate(atividades):
        grafo.add_node(atividade)
        for predecessor in predecessores[i].split(','):
            if predecessor != ' ':
                grafo.add_edge(predecessor, atividade)
    grafo_ = copy.deepcopy(grafo)
    caminho_critico = []
    while grafo_.nodes:
        for nodo in nx.topological_sort(grafo_):
            if not nx.descendants(grafo_, nodo):
                caminho_critico.append(nodo)
                grafo_.remove_node(nodo)
                break
    return caminho_critico, grafo
```

Só mostrando que é possivel resolver usando grafos, eu encontrei o mesmo caminho, só tive dificuldades para recuperar o tempo minimo para realização de cada atividade, porém é possivel, mas um trabalho atoa para mim aqui aagora.

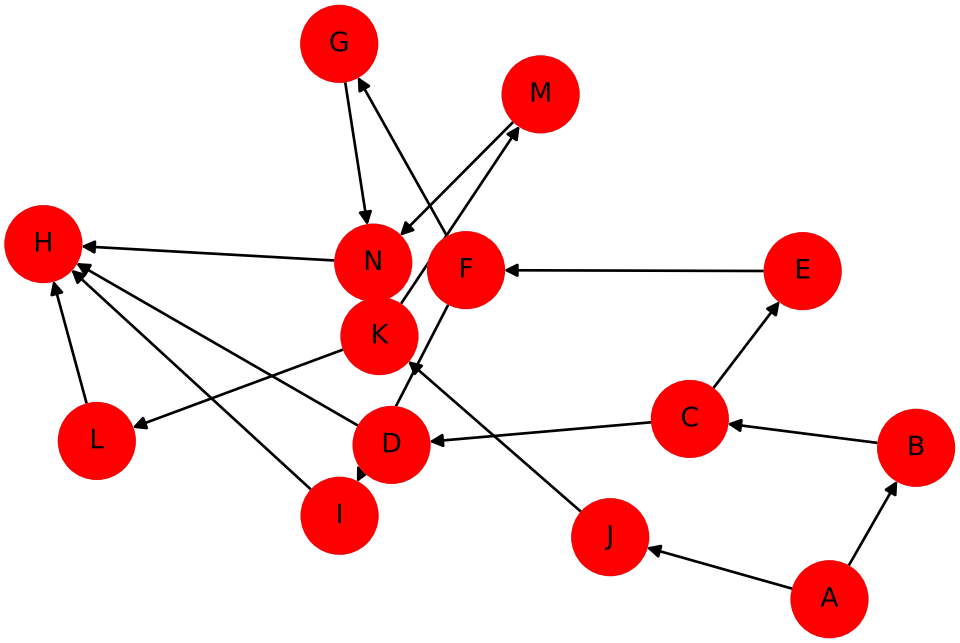
```python
import networkx as nx
import matplotlib.pyplot as plt

def plot_caminho_critico(grafo, caminho_critico):
    pos = nx.spring_layout(grafo)  # Define a posição dos nós (pode usar outros layouts tambe
    nx.draw(grafo, pos, with_labels=True, node_size=800, node_color='skyblue', font_size=10)

    # Destaca o caminho crítico com uma cor diferente (por exemplo, vermelho)
    nx.draw_networkx_nodes(grafo, pos, nodelist=caminho_critico, node_color='red', node_size=

    plt.title("Grafo do Caminho Crítico")
    plt.show()
```

Usando um modelo da "Literatura(sala de aula para ser mais preciso)"

```
def cmax(df):
    import pulp

    atividades      = df['Atividade'].tolist()
    duracoes        = df['Duração [Dias]'].tolist()
    predecessores   = df['Atividades Predecessoras'].tolist()

    problem = pulp.LpProblem("Minimize Cmax", pulp.LpMinimize)
    ti      = pulp.LpVariable.dicts("ti", atividades, lowBound=0, cat=pulp.LpContinuous)
    Cmax    = pulp.LpVariable("Cmax", lowBound=0, cat=pulp.LpContinuous)
    # (2)
    for i, atividade in enumerate(atividades):
        problem += ti[atividade] + duracoes[i] <= Cmax
        for predecessor in predecessores[i].split(','):
            if predecessor != " ":
                problem += ti[atividade] >= ti[predecessor] + duracoes[atividades.index(prede
    problem.solve()
    print(f"Custo caminho critico: {Cmax.varValue}")
    for i, atividade in enumerate(atividades):
        print(f"Custo para chegar em {atividade}: {ti[atividade].varValue}")
```

A vantagem é que a função objetivo retorna o valor de custo critico já

```
caminho_critico ,grafo = grafo_caminho_critico(df)
plot_caminho_critico(grafo, caminho_critico)

print('Caminho Crítico com metodo grafo')
for atividade in caminho_critico:
    print(atividade)
```




```
Caminho Crítico com metodo grafo
H
D
L
I
N
M
K
J
G
F
E
C
B
A
```

Aqui mostrando como o modelo com p.o, consigo recuperar o tempo para realização de cada atividade, então a partir dai eu consigo definir qual das atividades que detem o valor maximo, também as atividades que gargalam.

```
pulp  = caminho_critico_pulp(df)

print('Caminho Crítico usando P.O com pulp')
for atividade, tempo in pulp.items():
    print(f"{atividade}: {tempo:.2f} dias")
```

```
C:\Users\tastc\AppData\Local\Programs\Python\Python311\Lib\site-packages\pulp\pulp.py:1298: U
  warnings.warn("Spaces are not permitted in the name. Converted to '_'")
```

```
Caminho Crítico usando P.O com pulp
A: 0.00 dias
B: 25.00 dias
C: 46.00 dias
D: 69.00 dias
E: 69.00 dias
F: 82.00 dias
G: 84.00 dias
H: 116.00 dias
I: 84.00 dias
J: 25.00 dias
K: 32.00 dias
L: 40.00 dias
M: 40.00 dias
N: 86.00 dias
```

obs: os valores não estão considerando o tempo deles mesmo apenas de seus predecessores, considere como o tempo para chegar la

```
cmax(df)
```

```
Custo caminho critico: 118.0
Custo para chegar em A: 0.0
Custo para chegar em B: 25.0
Custo para chegar em C: 46.0
Custo para chegar em D: 69.0
Custo para chegar em E: 69.0
Custo para chegar em F: 82.0
Custo para chegar em G: 84.0
Custo para chegar em H: 116.0
Custo para chegar em I: 84.0
```

```
Custo para chegar em J: 25.0
Custo para chegar em K: 32.0
Custo para chegar em L: 40.0
Custo para chegar em M: 40.0
Custo para chegar em N: 86.0
```