```
# Yêu cầu 1: Nhập đồ thị từ file text và thực hiện tô màu cho đỉnh
G=[]
file = "/content/drive/MyDrive/THTTNT/dothi.txt"
# đọc đồ thị
try:
  fi = open(file, 'r') # Mở file txt và thực hiện đọc
  for dong in fi: # Duyệt từng dòng
    dong = dong.strip() # Strip() giúp bỏ đi kí tự cuối
    if dong:
      cot = [int(x) for x in dong.split()] # Duyệt từng ký tự và chyển thành vào danh sách cột
    G.append(cot) # thêm cột vào đồ thị
  print("Đã load xong đồ thị!")
  print("Số đỉnh:", len(G))
except FileNotFoundError:
  print(f"Lỗi: Không tìm thấy file '{file}'.")
# Nhập đỉnh
dinh = "ABCDEF" #Nhập tên các đỉnh của đồ thị tưng ứng với từng cột và dòng của file txt
t_ = {} # tạo một Dictionary

for i in range(len(G)):# hàm len giúp đếm số dòng, chạy từ i đến số lượng dòng
    t_[dinh[i]] = i #gán đỉnh cho từng dòng vd: t_["A"] = 0

# Tính bậc
bac = []
for i in range(len(G)): # Duyệt từng dòng
    bac.append(sum(G[i])) # Tính bậc = tổng từng dòng

# Danh sách màu
colorDict = {}
for i in range(len(G)): # Duyệt từng dòng
    colorDict[dinh[i]] = ["Blue", "Red", "Yellow", "Green"] # Cấp phát màu cho từng đỉnh VD: colorDict["A"] = ["Blue", "Red", "

# Sắp xếp các đỉnh theo thứ tự bậc giảm dần
sortedNode = [] # Tạo mảng để lưu các định đã sắp xếp
indeks = [] #Lưu vị  trí đỉnh được đưa vào sortedNode
for i in range(len(bac)):
    _max = -1
    idx = -1
    for j in range(len(bac)):
        if j not in indeks: # kiểm tra j có trong mảng lưu vị trí chưa
            if bac[j] > _max:
                _max = bac[j]
                idx = j
    indeks.append(idx)
    sortedNode.append(dinh[idx])

# xử lý tô màu
theSolution = {}
for n in sortedNode:
    # Lấy màu đầu tiên trong danh sách
    if len(colorDict[n]) > 0:
        setTheColor = colorDict[n][0]
        theSolution[n] = setTheColor

        # Xem xét các đỉnh kề để loại bỏ màu này khỏi danh sách màu của chúng
        current_node_index = t_[n]
        adjacentNode = G[current_node_index]

        for j in range(len(adjacentNode)):
            # Nếu là đỉnh kề
            if adjacentNode[j] == 1:
                neighbor_name = dinh[j]
                # Nếu màu vừa tô có trong danh sách màu của đỉnh kề, thì xóa nó đi
                if setTheColor in colorDict[neighbor_name]:
                    colorDict[neighbor_name].remove(setTheColor)
    else:
        theSolution[n] = "Không tìm thấy máu để tô"

print("-" * 20)
print("KẾT QUẢ TÔ MÀU:")
for t, w in sorted(theSolution.items()):
    print(f"Đỉnh {t} = {w}")
```

```
Đã load xong đồ thị!
Số đỉnh: 6
```

```
--------------------
KẾT QUẢ TÔ MÀU:
Đỉnh A = Yellow
Đỉnh B = Blue
Đỉnh C = Red
Đỉnh D = Yellow
Đỉnh E = Blue
Đỉnh F = Red
```

Nhấp đúp (hoặc nhấn Enter) để chỉnh sửa

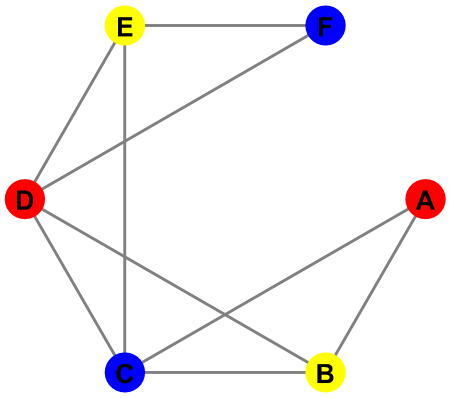
```
!pip install ColabTurtle # Tải thư viện
```

```
Collecting ColabTurtle
  Downloading ColabTurtle-2.1.0.tar.gz (6.8 kB)
  Preparing metadata (setup.py) ... done
Building wheels for collected packages: ColabTurtle
  Building wheel for ColabTurtle (setup.py) ... done
  Created wheel for ColabTurtle: filename=ColabTurtle-2.1.0-py3-none-any.whl size=7643 sha256=7171c6e83208510446d98678c07bb027c2
  Stored in directory: /root/.cache/pip/wheels/9f/af/64/ffd85f9858ed7d56b7293dcedbc9d461bf13c8cfc97e352bc8
Successfully built ColabTurtle
Installing collected packages: ColabTurtle
Successfully installed ColabTurtle-2.1.0
```

```
from ColabTurtle.Turtle import *
import math


node = "ABCDEF"
G = [
    [0, 1, 1, 0, 0, 0], # A
    [1, 0, 1, 1, 0, 0], # B
    [1, 1, 0, 1, 1, 0], # C
    [0, 1, 1, 0, 1, 1], # D
    [0, 0, 1, 1, 0, 1], # E
    [0, 0, 0, 1, 1, 0]  # F
]

print("Đã load dữ liệu!")

# tô màu đồ thị
t_ = {}
for i in range(len(G)):
    t_[node[i]] = i

bac = []
for i in range(len(G)):
    bac.append(sum(G[i]))

colorDict = {}
for i in range(len(G)):
    colorDict[node[i]] = ["Blue", "Red", "Yellow", "Green"]

sortedNode = []
indeks = []

for i in range(len(bac)):
    _max = -1
    idx = -1
    for j in range(len(bac)):
        if j not in indeks:
            if bac[j] > _max:
                _max = bac[j]
                idx = j
    indeks.append(idx)
    sortedNode.append(node[idx])

theSolution = {}
for n in sortedNode:
    if len(colorDict[n]) > 0:
        setTheColor = colorDict[n][0]
        theSolution[n] = setTheColor
        current_node_index = t_[n]
        adjacentNode = G[current_node_index]
        for j in range(len(adjacentNode)):
            if adjacentNode[j] == 1:
```

```
                neighbor_name = node[j]
                if setTheColor in colorDict[neighbor_name]:
                    colorDict[neighbor_name].remove(setTheColor)
    else:
        theSolution[n] = "No Color Available"

print("KẾT QUẢ TÔ MÀU:")
for t, w in sorted(theSolution.items()):
    print(f"Đỉnh {t} = {w}")

initializeTurtle() #Khởi tạo màn hình vẽ của ColabTurtle

speed(5)  # Chỉnh tốc độ con rùa
bgcolor("white") # Đặt màu nền
pensize(2) # Độ dày đường vẽ

center_x = 400 #Nhập tọa độ x, y
center_y = 250
radius = 150 # Bán kình tâm vòng tròn
positions = {} # giúp lưu tọa độ x,y của từng đỉnh
num_nodes = len(G) # Đếm số dòng (đỉnh) của đồ thị
angle_step = 360 / num_nodes

# Tính toạ độ
for i in range(num_nodes):
    angle_rad = math.radians(i * angle_step) # Đổi độ sang radian
    #Tìm đỉnh trên đường tròn
    x = center_x + math.cos(angle_rad) * radius
    y = center_y + math.sin(angle_rad) * radius
    positions[node[i]] = (x, y) # Lưu tọa độ đỉnh VD: positions["A"] = ()
# Vẽ cạnh
color("gray") # đổi màu cho đường vẽ
for i in range(len(G)):
    start_node = node[i] # Lấy đỉnh đầu VD: start_node = A
    start_pos = positions[start_node] # Lấy tọa độ đỉnh
    for j in range(i + 1, len(G)): # Duyệt các đỉnh còn lại
        if G[i][j] == 1: # Kiểm tra nó có cạnh(1) không
            end_node = node[j] # Lây đỉnh đó
            end_pos = positions[end_node] # Lấy tọa độ của đỉnh đó
            # Thực hiện vẽ
            penup()# Nhấc búp lên
            goto(start_pos[0], start_pos[1]) # Đặt điểm bắt đầu
            pendown() #Đặt bút xuống và vẽ
            goto(end_pos[0], end_pos[1]) # Vẽ đến đỉnh kết thúc

# Vẽ đỉnh và tô màu
for n in node: # Duyệt từng đỉnh (từng dòng)
    x, y = positions[n] # Lấy tọa độ đỉnh
    c = theSolution.get(n, "white") # Lấy màu từ theSolution nếu không có thì mặc định là trắng
    if c == "No Color Available": c = "gray"

    penup() #Nhấc bút
    goto(x, y) # Đi tới đỉnh
    color(c.lower()) # Tô Màu tương ứng

    # Vẽ chấm tròn
    old_size = pensize()
    pensize(30)
    pendown()
    forward(0)
    penup()
    pensize(old_size)

    # Viết tên
    goto(x, y + 8)
    color("black")
    write(n, align="center", font=(20, "Arial", "bold"))

goto(50, 450)
```

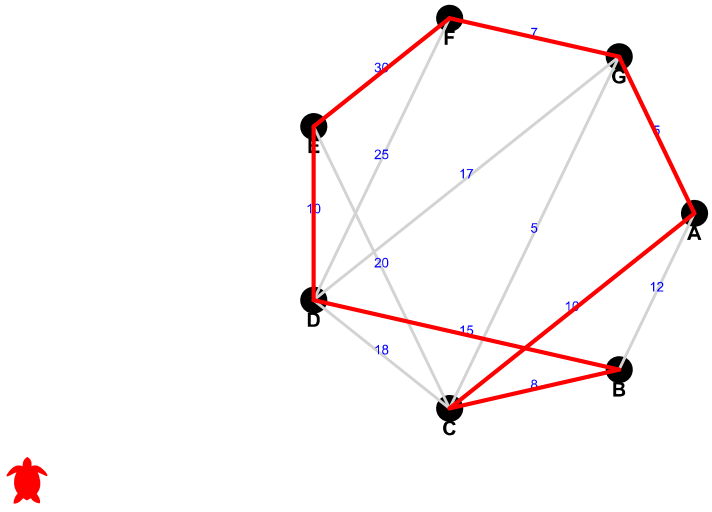
```
Đã load dữ liệu!
KẾT QUẢ TÔ MÀU:
Đỉnh A = Red
Đỉnh B = Yellow
Đỉnh C = Blue
Đỉnh D = Red
Đỉnh E = Yellow
Đỉnh F = Blue
```




```
from ColabTurtle.Turtle import *
import math
import itertools # Thư viện dùng để sinh ra các hoán vị (tất cả các cách sắp xếp)

node = "ABCDEFG"

# Ma trận kề trọng số (Weighted Matrix)
# Hàng là điểm đi, Cột là điểm đến. Giá trị là khoảng cách (km).
G = [
    [0,  12, 10, 0,  0,  0 , 5, 13], # A (0)
    [12, 0,  8,  15, 0,  0, 0], # B (1)
    [10, 8,  0,  18, 20, 0, 5], # C (2)
    [0,  15, 18, 0,  10, 25, 17], # D (3)
    [0,  0,  20, 10, 0,  30, 0], # E (4)
    [0,  0,  0,  25, 30, 0, 7],  # F (5)
    [13,  0,  5,  17, 0, 7, 0]
]
# INF (Infinity): Đại diện cho vô cực, dùng cho các cặp điểm không có đường đi
INF = float('inf')
num_nodes = len(G)

initializeTurtle()
speed(13)
bgcolor("white")
pensize(2)


center_x = 400   # Tâm vòng tròn (ngang)
center_y = 250   # Tâm vòng tròn (dọc)
radius = 150
positions = {}   # Dictionary để lưu tọa độ x,y của từng đỉnh
angle_step = 360 / num_nodes # Góc lệch giữa các đỉnh (360 chia 6 = 60 độ)


for i in range(num_nodes):
    angle_rad = math.radians(i * angle_step) # Đổi độ sang Radian
    x = center_x + math.cos(angle_rad) * radius # Tính x
    y = center_y + math.sin(angle_rad) * radius # Tính y
    positions[i] = (x, y)

# VẼ ĐỒ THỊ
```

```
print("Đang dựng bản đồ (Vẽ nhanh)...")

# 3.1 Vẽ các Đỉnh (Thành phố)
for i in range(num_nodes):
    x, y = positions[i]
    penup()      # Nhấc bút
    goto(x, y)   # đến tọa độ đỉnh


    color("black")
    old_size = pensize()
    pensize(20)
    pendown()
    forward(0)
    penup()
    pensize(old_size)

    # Viết tên thành phố (A, B, C...)
    goto(x, y + 20)      # Đến tọa độ đỉnh
    write(node[i], align="center", font=(15, "Arial", "bold"))

#Vẽ các Cạnh
color("lightgray") # Màu xám nhạt cho đường nền
for i in range(num_nodes):
    for j in range(i + 1, num_nodes): # j chạy từ i+1 để không vẽ lặp lại đường cũ
        if G[i][j] > 0: # Nếu khoảng cách > 0 tức là có đường đi
            start = positions[i]
            end = positions[j]

            # Vẽ dây nối
            penup()
            goto(start)
            pendown()
            goto(end)


            mid_x = (start[0] + end[0]) / 2
            mid_y = (start[1] + end[1]) / 2
            penup()
            goto(mid_x, mid_y)
            color("blue") # Số màu xanh
            write(str(G[i][j]), align="center", font=(10, "Arial", "normal"))
            color("lightgray") # Trả lại màu xám để vẽ tiếp dây khác

# THUẬT TOÁN TSP (BRUTE FORCE - VÉT CẠN) ---
print("-" * 30)
print("Đã vẽ xong! Đang tính toán lộ trình tối ưu...")

# Hàm tính tổng quãng đường của một lộ trình cụ thể
def tinh_tong_quang_duong(lo_trinh):
    tong_kc = 0
    # Duyệt qua từng chặng trong lộ trình
    for i in range(len(lo_trinh) - 1):
        u = lo_trinh[i]   # Điểm đi
        v = lo_trinh[i+1] # Điểm đến
        khoang_cach = G[u][v]

        # Nếu gặp đoạn đường cụt (giá trị 0), trả về Vô cực ngay
        if khoang_cach == 0: return INF
        tong_kc += khoang_cach
    return tong_kc

# cố định điểm đầu là A (0), chỉ hoán vị các điểm còn lại (1,2,3,4,5)
dinh_con_lai = list(range(1, num_nodes))
tat_ca_lo_trinh = itertools.permutations(dinh_con_lai)

duong_di_ngan_nhat = []
khoang_cach_ngan_nhat = INF
dem_so_truong_hop = 0 # Biến đếm để biết đã thử bao nhiêu cách

# Duyệt qua tất cả các trường hợp có thể xảy ra
for phuong_an in tat_ca_lo_trinh:
    dem_so_truong_hop += 1
    # Tạo lộ trình khép kín: 0 -> [các điểm giữa] -> 0
    lo_trinh_hien_tai = [0] + list(phuong_an) + [0]

    do_dai = tinh_tong_quang_duong(lo_trinh_hien_tai)
```

```
    # Cập nhật nếu tìm thấy đường ngắn hơn
    if do_dai < khoang_cach_ngan_nhat:
        khoang_cach_ngan_nhat = do_dai
        duong_di_ngan_nhat = lo_trinh_hien_tai

# In kết quả tính toán
print(f"Đã kiểm tra tổng cộng: {dem_so_truong_hop} trường hợp.")
ten_duong_di = [node[i] for i in duong_di_ngan_nhat]
print(f"-> LỘ TRÌNH TỐI ƯU: {' -> '.join(ten_duong_di)}")
print(f"-> TỔNG QUÃNG ĐƯỜNG: {khoang_cach_ngan_nhat} km")

#  MÔ PHỎNG SHIPPER DI CHUYỂN
if khoang_cach_ngan_nhat != INF:
    print("Bắt đầu mô phỏng giao hàng...")
    speed(3)
    color("red")     # Đổi bút màu đỏ
    pensize(3)       # Tăng độ đậm nét vẽ

    # Di chuyển rùa đến điểm xuất phát đầu tiên
    start_point = positions[duong_di_ngan_nhat[0]]
    penup()
    goto(start_point)
    pendown()

    # [NOTE] Vòng lặp vẽ từng chặng đường
    for i in range(1, len(duong_di_ngan_nhat)):
        next_node_idx = duong_di_ngan_nhat[i] # Lấy chỉ số điểm tiếp theo
        next_pos = positions[next_node_idx]   # Lấy tọa độ điểm tiếp theo
        goto(next_pos)                        # Rùa chạy đến đó

    print("Giao hàng hoàn tất!")
else:
    print("Không tìm được đường đi kín nào!")

# Đưa rùa ra chỗ khác cho đỡ che hình
penup()
goto(50, 450)
```




```
Đang dựng bản đồ (Vẽ nhanh)...
------------------------------
Đã vẽ xong! Đang tính toán lộ trình tối ưu...
Đã kiểm tra tổng cộng: 720 trường hợp.
-> LỘ TRÌNH TỐI ƯU: A -> G -> F -> E -> D -> B -> C -> A
-> TỔNG QUÃNG ĐƯỜNG: 85 km
Bắt đầu mô phỏng giao hàng...
Giao hàng hoàn tất!
```