```python
# -*- coding: utf-8 -*-
"""
Created on Thu Feb  2 01:26:05 2023

@author: dconduche
"""


"""LARGEUR"""
import collections as co


def parcours_en_largeur(G, s):
    n = len(G)
    # initialisation
    c = ['b'] * n
    f = co.deque()
    p = [-1] * n
    d = [-1] * n
    # le sommet de départ :
    f.append(s)
    c[s] = 'g'
    d[s] = 0
    while len(f) > 0:
        u = f.popleft()  # On prend le sommet arrivé en premier dans la queue
        for a in G[u]:   # on parcourt les sommets voisins de u..
            if c[a] == 'b':  # ..blancs
                c[a] = 'g'    # on colorie a en gris
                f.append(a)  # on le rajoute à la « liste des gris »
                p[a] = u     # son père est u
                d[a] = d[u] + 1  # La distance dist(s,a) est dist(s,u) + 1 arête
        c[u] = 'n'       # on a fini de visiter u (et les arêtes partant de u)
    return (p, d)


"""PROFONDEUR"""


def parcours_en_profondeur(G, s):
    n = len(G)
    # initialisation
    c = ['b'] * n
    p = [-1] * n

    def visiter_sommet(u):
        c[u] = 'g'          # on colorie u en gris
        for a in G[u]:      # on parcourt les sommets voisins de u..
            if c[a] == 'b':  # ..blancs
                p[a] = u     # son père est u
                visiter_sommet(a)
        c[u] = 'n'          # on a fini de visiter u (et les arêtes partant de u)
    # le sommet de départ :
    visiter_sommet(s)
    return p

"""CHEMIN DANS UN GRAPHE à partir de la liste des pères"""


def chemin(u, p):
    L = []

    def cheminRec(u):
        if u != -1:
            L.append(u)
            cheminRec(p[u])
    cheminRec(u)
    L.reverse()
    return L


"""DIJKSTRA"""


def relacher(u, a, poids, d, p):
    if d[a] == 'inf' or d[a] > d[u] + poids:
        d[a] = d[u] + poids
        p[a] = u


def choix_sommet(d, E):
    s0 = E[0]
    dmin = d[s0]
    for s in E[1:]:
        if dmin > d[s]:
            s0 = s
            dmin = d[s0]
    E.remove(s0)
    return s0


def dijkstra(G, s):
    # Initialisation
    n = len(G)
    d = ['inf'] * n
    p = [-1] * n
    E = []
    # Initialisation : s
    E.append(s)
    d[s] = 0
    while len(E) > 0:
        u = choix_sommet(d, E)  # On laisse choix_sommet choisir le sommet.
        for a, poids in G[u]:   # On parcourt les sommets voisins de u.
            if d[a] == 'inf':  # S'il est blanc,
                E.append(a)  # on le rajoute à la « liste des gris »
            if a in E:  # S'il est gris (désormais), ou l'était,
                relacher(u, a, poids, d, p)  # père et distance : relacher
    return (p, d)


"""Seules modifications pour dijkstra_cible : sortir si on a atteint sc via
choix_sommet. Et retourner False si on sort du while sans avoir trouvé sc."""


def dijkstra_cible(G, s, sc):
    # Initialisation
    n = len(G)
    d = ['inf'] * n
    p = [-1] * n
    E = []
    # Initialisation : s
    E.append(s)
    d[s] = 0
    while len(E) > 0:
        u = choix_sommet(d, E)
        if u == sc:        # Si on s'apprête à colorier en noir sc,
            return p, d    # alors on trouvé le plus court chemin s->sc : sortir
        for a, poids in G[u]:
            if d[a] == 'inf':
                E.append(a)
            if a in E:
                relacher(u, a, poids, d, p)
    return False


"""A étoile"""
"""Adapter Dijkstra à la nouvelle syntaxe par dictionnaire.
Seule modification algorithmique : choix_sommet devient choix_sommet_Ae"""


def dist(a, b):
    return ((a[0]-b[0])**2+(a[1]-b[1])**2)**.5


def choix_sommet_Ae(d, E, sc):
    s0 = E[0]
    dmin = d[s0] + dist(s0, sc)
    for s in E[1:]:
        distance = d[s] + dist(s, sc)
        if dmin > distance:
            s0 = s
            dmin = distance
    E.remove(s0)
    return s0


def Aetoile(G, s, sc):
    # Initialisation : désormais, des dictionnaires vide, c'est plus simple !
    d = {s: 'inf' for s in G.keys()}
    p = {}
    E = []
    # Initialisation : s
    E.append(s)
    d[s] = 0
    while len(E) > 0:
        u = choix_sommet_Ae(d, E, sc)  # seule modification
        if u == sc:
            return p, d
        for a in G[u]:
            poids = dist(a, u)
            if d[a] == 'inf':
                E.append(a)
            if a in E:
                relacher(u, a, poids, d, p)
    return False
```