Ch 21bis. Graphe - le plus court chemin - Algorithme A Star.

1 L'algorithme A*

```
### les voisins sont les cases àcôté du point considéré en ligne droite ou en diagonale
def estVoisin(M:dict,pt1:tuple):
    l,c=pt1
    voisins=[]
    for i in [-1,0,1]:
        for j in [-1,0,1]:
            if (l+i,c+j) in M:
                voisins.append((l+i,c+j))
    voisins.remove(pt1)
    return voisins

# >>> estVoisin(G,[1,2])
# [[0, 1], [0, 2], [0, 3], [1, 1], [1, 3], [2, 1], [2, 2], [2, 3]]

### on peut calculer la distance entre deux points voisins en dehors de Astar
def distance(pt1:tuple,pt2:tuple):
    '''pt1 et pt2 doivent être voisin en ligne ou en diagonale
    '''
    ligne=0
    colonne=0
    i,j=pt1
    l,c=pt2
    if not i==l:
        ligne=1
    if not j==c:
        colonne=1
    if colonne==ligne:
        return 14
    else:
        return 10

# >>> distance([1,2],[0, 3])
# 14



def Astar(M:dict,depart,fin,visited=[]):
    '''calcul le plus court chemin en partant de départ pour atteindre
    arrivée par l'algorithme de Astar avec un heuristique de distance la plus courte
```

```
    entrées :
    M : dict, dictionnaire dont chaque sommet est un couple de coordonnées et sa valeur une
        liste de 4 éléments : G, H, F et le prédécesseur
    départ : un sommet de M dont on connait les coordonnées
    fin : un sommet de M dont on connait les coordonnées
    sortie : None (ne renvoie rien)
    '''
    if depart not in list(M.keys()):
        raise TypeError('Le sommet de départ n\'est pas dans le graphe')
    if fin not in list(M.keys()):
        raise TypeError('Le sommet d\'arrivée n\'est pas dans le graphe')
    # condition de sortie de la boucle récursive
    if depart==fin:
        # on construit le plus court chemin et on l'affiche
        chemin=[]
        pred=fin
        while pred != None:
            chemin.append(M[pred][-1])
            pred=M[pred][-1]
        chemin.reverse() # on retourne la liste dans le bon ordre
        print ('chemin le plus court :'+str(chemin)+' cout='+str(M[fin][2]))

    else :
        # au premier passage, on initialise le coÃ»t à0
        if visited==[] :
            M[depart][0]=0 # changement
        # on visite les successeurs de depart
        voisins=estVoisin(M,depart) # changement
        for voisin in voisins:
            if voisin not in visited:
                # heuristique
                G = M[depart][0] + distance(depart,voisin)
                H = heuristique(voisin,fin)
                F = G + H

                if F < M[voisin][2]:
                    # les distances calculées
                    M[voisin][0] = G
                    M[voisin][1] = H
                    M[voisin][2] = F
                    # le prédécesseur
                    M[voisin][3] = depart
        # On marque comme "visited"
        visited.append(depart)
        # Une fois que tous les successeurs ont été visités : récursivité
        # On choisit les sommets non visités avec la distance globale la plus courte
        # On ré-exécute récursivement Astar en prenant depart='x'
```

```
        not_visited={}
        for k in M.keys():
            if not k in visited :
                not_visited[k] = M[k][2]
        x=min(not_visited, key=not_visited.get)
        Astar(M,x,fin,visited)
```