DM 4 – Algorithme glouton du plus court chemin.

Réflexions autour de la résolution du problème par force brute

Question 1 *Dénombrer le nombre de chemins possibles démarrant par A et passant par 3 points.*

Corrigé

6 chemins ($A-P_1-P_2-P_3$, $A-P_1-P_3-P_2$, $A-P_2-P_1-P_3$, $A-P_2-P_3-P_1$, $A-P_3-P_1-P_2$ et $A-P_3-P_2-P_1$).

Question 2 *Dénombrer le nombre de chemins possibles démarrant par A et passant par* 10 *points.*

Corrigé

$n!$

Question 3 *Conclure sur le nombre d'opérations (et donc sur le temps de calcul) nécessitant de déterminer la distance de l'ensemble des chemins passant par n points.*

Corrigé

Beaucoup de temps !

Résolution du problème

Question 4 *Écrire la fonction* `distance(p1:list,p2:list)-> float` *permettant de calculer la distance euclidienne entre deux points. Vous importerez les fonctions que vous jugerez utile. La fonction* `test_Q4()` *permet de valider votre fonction dans un cas.*

Corrigé

```
from math import sqrt
def distance(p1, p2):
    x1, y1 = p1
    x2, y2 = p2
    return sqrt((x1 - x2) ** 2 + (y1 - y2) ** 2)

def distance2(p1, p2): # sans import
    x1, y1 = p1
    x2, y2 = p2
    return ((x1 - x2) ** 2 + (y1 - y2) ** 2)**0.5
```

Question 5 *Écrire la fonction* `distances(a:list,pts:list)-> list` *permettant de calculer l'ensemble des distances entre chacun des n points P de la liste* `pts` *et chacune des distances entre le point* `a` *et chacun des points de la listes* `pts` *conformément à l'exemple ci-dessus.*

Corrigé

```
def distances(pts, dep):
    n = len(pts)
    tab = [(n+1)*[0] for i in range(n+1)]
    for i in range(n):
        for j in range(i):
            tab[i][j] = distance(pts[i], pts[j])
            tab[j][i] = tab[i][j]
        tab[n][i] = distance(dep, pts[i])
        tab[i][n] = tab[n][i]
    return tab
```

Question 6 *En prenant compte de la note précédente, estimer le nombre de distances à calculer par l'algorithme. (Réaliser l'application numérique poiy*

Corrigé

Le tableau des distances contient $n+1$ lignes et colonnes soit $(n+1)\times(n+1)$ valeurs.
Les termes de la diagonale sont tous nuls. Il ne faut donc pas les calculer. Il y en a $n+1$.
Le tableau est symértrique ce qui signifie que `tab[i][j]=tab[j][i]`.
Il faut donc calculer $\dfrac{(n+1)^2-(n+1)}{2}=\dfrac{n(n+1)}{2}$ distances.

Soit un chemin, quelconque passant par un ensemble de points contenu dans le tableau des distances. Pour modéliser ce chemin, on utilise la liste des index des points qui le constituent. Ainsi si `chemin = [n,4,0,n-1]` alors c'est qu'on a parcouru le chemin $A \to P_5 \to P_1 \to P_n$.

Question 7 *Écrire la fonction `longueur(chemin:list, tab:list) -> float` où `tab` est un tableau de distances déterminé par la fonction `distances`.*

Corrigé

```
def longueur(chemin, dist):
    d = 0
    id_pt = len(dist) - 1
    for point in chemin:
        d = d + dist[id_pt][point]
        id_pt = point
    return d
```

Question 8 *Proposer une version récursive de cet algorithme. On la notera `longueur_rec(chemin:list, tab:list) -> float`.*

Corrigé

```
def longueur(chemin, dist): A TESTER !!
    d = 0
    if len(chemin) == 2 :
        return dist[chemin[0][chemin[1]]
    else :
        return d=d + longueur(chemin([1:],dist))
```

Corrigé

Question 9 *On considère que* `dispo=[True, True, False, False, True, False]`*. Combien existerait-il de points dans la variable* `pts` *définie précédemment ? Combien de points ont été parcouru ? Citer les points visités.*

Corrigé

`pts` est constitué de 5 points. 3 points ont été visités : A, P_3 et P_4.

Question 10 *Dans la figure ci-dessus, expliquer pourquoi il y a une case à ne pas visiter ?*

Corrigé

On ne visite pas la case où on est ...

Question 11 *Écrire une fonction* `indice(i:int, tab:list, dispo:list)->int` *permettant de déterminer l'index du point le plus proche du point d'index* `i`*, parmi les points disponibles.* `tab` *désigne le tableau des distances créé avec la fonction* `distances`*.*

Corrigé

```
def indice(position, dist, dispo):
    n = len(dist) - 1
    global dim
    mini = 3 * dim # supérieur àla diagonale du carré
    for i in range(n):
        if dispo[i]:
            d = dist[position][i]
            if d < mini:
                mini = d
                ind = i
    return ind
```

Question 12 *Écrire la fonction* `plus_court_chemin(dist:list)->list` *permettant de construire la chemin le plus court. Le chemin sera constitué de la liste des index des points. On initialisera donc le chemin avec le plus grand index de* `dist`*. On initialisera la liste* `dispo` *des points disponibles. Á chaque itération, on ajoutera à* `chemin` *le point le plus proche puis on mettra à jour la variable* `dispo`*.*

Corrigé

```
def plus_court_chemin(dist):
    n = len(dist) - 1
    chemin = []
    dispo = n * [True]
    position = n
    while len(chemin) < n:
        position = indice(position, dist, dispo)
        chemin.append(position)
```

Corrigé

```
        dispo[position] = False
    return chemin
```

Représentation du chemin

Question 13 *Tester le fonction `plot_chemin()`. Commenter l'ensemble des lignes en effectuant les regroupements vous paraissant nécessaires.*