NSI : Projet Koh-Lanta (niveau 2)

par G. Coste

Contexte :

Dans l'émission Koh-Lanta, il y a évidemment l'épreuve mythique des poteaux mais il y a également une belle épreuve (d'immunité ou de confort) qui met en jeu *plusieurs* poteaux !

Chaque aventurier débute perché au sommet d'un poteau (le *poteau de départ, souvent nommé 'A'*). Il dispose d'une poutre (de longueur L) qu'il peut déplacer et positionner en appui sur 2 poteaux (celui sur lequel il se trouve et celui sur lequel il souhaite aller) à condition qu'ils soient suffisamment proches (leur écartement doit être inférieur ou égal à L). En équilibre sur la poutre, l'aventurier doit passer sur l'autre poteau sans tomber (sinon, il recommence au *poteau de départ*).

Le but de l'épreuve est d'atteindre en premier, en passant de poteau en poteau, le *poteau d'arrivée* (souvent nommé 'Z'). Pour cela, l'aventurier a intérêt à minimiser le nombre de "traversées". En effet, la distance parcourue importe peu car c'est le fait de déplacer la poutre qui lui fait perdre du temps.

L'épreuve est donc définie par :

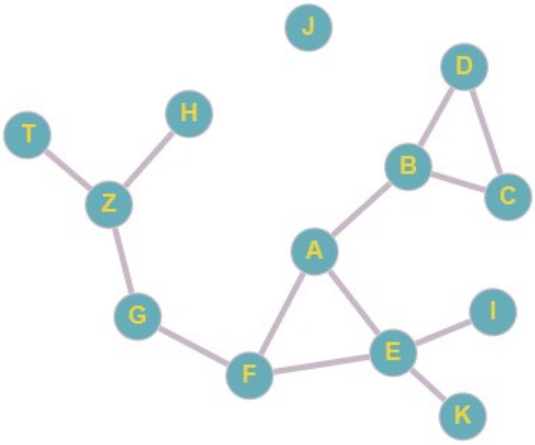
- un ensemble de `N` poteaux (tous ne servent pas forcément) dont on connait la position (x ; y).
- une poutre de longueur L
- un poteau de départ
- un poteau d'arrivée

Objectifs :

Les poteaux de l'épreuve sont assimilés aux sommets d'un graphe non orienté (on peut effectuer la traversée dans les 2 sens) et non pondéré. Deux sommets sont reliés par une arrête si la distance qui les sépare est inférieure ou égale à la longueur de la poutre : on dit alors que le poteau est accessible à partir de l'autre poteau.

Le but de ce projet est d'identifier le chemin (ou les chemins) le plus court (le moins de traversées possible) pour aller du poteau de départ jusqu'au poteau d'arrivée, en utilisant la poutre de longueur L.

Voici un exemple d'un tel *graphe* :



Dans cet exemple, le chemin le plus court de 'A' (poteau de départ) vers 'Z' (poteau d'arrivée) est :
'A', 'F', 'G', 'Z'

Remarque : Le tracé du graphe ne présume en rien de la position *géographique* des poteaux. Il matérialise uniquement le fait que certains poteaux sont accessibles à partir d'autres poteaux.

Dans cet exemple, le poteau 'J' n'est accessible depuis aucun autre poteau (le graphe est *non connexe*).

Programmation Orientée Objet (POO)

Chaque poteau est une instance de la classe `Poteau` caractérisée par les attributs :

- `nom` dont la valeur est une chaine de caractères (`str`) : 'A' ou 'B' ou 'C'...
- `x` dont la valeur, *abscisse* du poteau, est un nombre (`float`)
- `y` dont la valeur, *ordonnée* du poteau, est un nombre (`float`)

Une épreuve est une instance de la classe `Epreuve` caractérisée par les attributs :

- une liste de poteaux, instances de la classe `Poteau` (tous ne servent pas forcément) dont on connait le nom et la position (x ; y).
- une poutre de longueur L
- un poteau de départ, instance de la classe Poteau
- un poteau d'arrivée, instance de la classe Poteau

La liste des poteaux est obtenue à l'aide de la lecture d'un fichier texte dans lequel chaque ligne est de la forme : "nom_du_poteau, abscisse_du_poteau, ordonnée_du_poteau"
Le premier poteau de cette liste sera le poteau de départ.
Le dernier poteau de cette liste sera le poteau d'arrivée.

C'est à vous !

La rédaction des *docstrings* et les *doctests* est attendue.

Définir la classe `Poteau` : constructeur, attributs, accesseurs (les mutateurs ne sont pas nécessaires)
Implémenter les méthodes :

- `__eq__(self, autre_poteau)` qui surcharge la méthode d'égalité : 2 poteaux sont identiques s'ils ont le même nom, la même abscisse et la même ordonnées.
- `distance(self, autre_poteau)` : qui retourne la distance géométrique séparant le poteau autoréférencé et l'autre poteau passé en paramètre
  Rappel : la distance géométrique entre 2 points A et B est $AB=\sqrt{(x_A-x_B)^2+(y_A-y_B)^2}$
- `est_accessible_a_partir(self, autre_poteau, d)` : qui retourne `True` si le poteau autoréférencé est accessible depuis l'autre poteau passé en paramètre(et inversement) à l'aide d'une poutre de longueur `d`. `False` sinon.
- `liste_de_poteaux_accessibles(self, liste_poteaux, d)` : qui retourne la liste des poteaux accessibles (parmi la liste de poteaux passée en paramètre) depuis le poteau autoréférencé à l'aide d'une poudre de longueur `d`.

Définir la classe `Epreuve` : constructeur, attributs, accesseurs et mutateurs
On pourra envisager un constructeur ayant pour paramètre :

- `L` la longueur de la poutre
- Le nom d'un fichier .txt dont la lecture génèrera les autres attributs :
  - La liste des poteaux dont on connaît la position
  - Le poteau de départ
  - Le poteau d'arrivée

Ex: `>>> epreuve_de_confort = Epreuve(3, "fichier1.txt")`

`Aide pour lire un fichier txt :`

```
def lire_fichier_en_tuples(nom_fichier):
    tuples_liste = []
    with open(nom_fichier, 'r') as fichier:
        for ligne in fichier:
            elements = ligne.split(',')
            lettre = str(elements[0])
            valeur1 = int(elements[1])
            valeur2 = int(elements[2])
            tuples_liste.append((lettre, valeur1, valeur2))
    return tuples_liste
```

Implémenter les méthodes :

- `plus_court_chemin(self)` : qui retourne la liste des noms des poteaux parcourus depuis le poteau de départ, jusqu'au poteau d'arrivée, qui minimise le nombre de déplacements (il s'agit alors d'un parcours en ..... (*largeur* ou *profondeur* ???) du graphe, ce qui nécessite l'utilisation da la classe ..... (`Pile` ou `File` ???) dont il faudra importer le module.

Dans l'exemple étudié : `['A', 'F', 'G', 'Z']`

- `nombre_de_traversees(self)`qui retourne le nombre minimal d'arrêtes parcourues lorsqu'on emprunte le plus court chemin depuis le poteau de départ, jusqu'au poteau d'arrivée. Cette méthode utilisera la méthode `plus_court_chemin`.

  Dans l'exemple étudié : `3`