TD TNSI – Graphe algorithme de Dijkstra

Calcul du plus court chemin

Q1 Recopier le code de l'algorithme de Dijkstra suivant

Soit G un graphe connexe pondéré dont tous les poids sont positifs. On nomme {S1, ..., Sn} la liste de ses sommets. On recherche une plus courte chaine reliant S1, à Sn.

- Définition des graphes utilisés :
- Graphes pondérés sous forme de dictionnaires exemple :

```
G = {
'A':{'B':3, 'E':3,'C':3,'D':2},
'B':{'A':3, 'C':5, 'D':4, 'E':2},
'C':{'A':3, 'B':5, 'D':2, 'S':2},
'D':{'A':2,'B':4, 'C':2, 'S':2},
'E':{'A':3, 'B':2},
'S':{'C':2, 'D':2},
}

graphe = G
```

- **Initialisation :**
  - On affecte S1, du poids 0
  - On affecte tous les autres sommets d'un poids infini (∞).

```
from math import inf # permet d'utiliser + infini
depart = 'E'
arrivee = 'S'
# initialisation
marque = {}
#marque dictionnaire pour affecter une distance aux sommets explorés
for sommet in graphe:
    marque[sommet] = inf
#le sommet de départ est affecté avec une distance 0
marque[depart] = 0
```

  - On crée un dictionnaire pour garder en mémoire les sommets non encore traités

```
#création d'une liste des sommets non encore visités
non_selectionnes = [sommet for sommet in graphe]
print(non_selectionnes)
```

  - Création d'un dictionnaire pour pouvoir remonter le chemin parcouru

```
#création d'un dictionnaire sommet distance à parcourir de l'origine
pere = {}
pere[depart] = None
```

TD TNSI – Graphe algorithme de Dijkstra

Calcul du plus court chemin

- **Itération :**

TANT QU' il reste des sommets Sk non sélectionnés :

Sélectionner, parmi les sommets non-sélectionnés, le sommet SA ayant la marque la plus petite.

Pour chaque sommet SB adjacent à SA et non déjà sélectionné :

Calculer p = (marque de SA) + (poids de l'arête A-B).

SI p < marque de SB

ALORS remplacer marque de SB par p

SINON conserver marque de SB.

FIN TANT QUE

```
#boucle principale parcours des différents sommets non encore parcourus :

# jusqu'à
while non_selectionnes:
    # sélection:
    marquePlusPetite = inf
    #détermination du sommet avec la distance la plus petite
    for s in non_selectionnes:
        if marque[s] < marquePlusPetite:
            marquePlusPetite = marque[s]
            sommetPlusPetit = s
        print(s)
    # cas où on arrive sur le sommet à l'arrivée
    if sommetPlusPetit == arrivee:
        break
    # on retire le sommet avec la distance la plus petite
    non_selectionnes.remove(sommetPlusPetit)
    # mise à jour des voisins du sommet sélectionné:
    VoisinsAVisiter = [sommet for sommet in graphe[sommetPlusPetit] if sommet
    in non_selectionnes]
    for sommet in VoisinsAVisiter:
            p = marque[sommetPlusPetit] + graphe[sommetPlusPetit][sommet]
            if p < marque[sommet]:
                marque[sommet] = p
                pere[sommet] = sommetPlusPetit
```

- **Exploitation des résultats**

```
# affichage de la distance
print("La distance de {} à {} est de longueur {}.".format(depart, arrivee,
marque[arrivee]))
# affichage du chemin
chemin = arrivee
sommet = arrivee
while pere[sommet] != None:
    chemin = pere[sommet] +' - '+ chemin
    sommet = pere[sommet]

print("Le chemin de {} à {}: {}.".format(depart, arrivee,chemin))
```

Q2 Tester ce code

Q3 Inspirez vous de ce code pour ajouter une méthode « disjtra » à l'objet graphe du tp précédent. On peut décomposer en plusieurs méthodes :

def initial_disjtra(self) :

def disjtra(self, depart, arrivee):