TP_Python_SRI_2

September 13, 2021

1 Programmation objet : les graphes

1.1 Echauffement

Pour commencer, programmez et testez une classe Stack avec un comportement de pile (dernier entré - premier sorti), en utilisant une liste pour stocker les éléments (indice: les listes ont déjà une méthode pop, et sont optimisées pour ajouter des éléments à la fin de la liste).

1.2 Graphes

Définissez maintenant une classe Graph qui implémente un graphe, dirigé ou non. Il devra inclure au moins les méthodes suivantes˜:

- depth_first: qui itère sur les noeuds en profondeur d'abord, à partir d'un noeud donné
- comp_con: qui donne les composantes connexes (à vous de décider du type renvoyé)
- path: qui donne le plus court chemin d'un noeud à un autre (en nombre d'arcs/arête), par exemple avec l'algorithme de Dijsktra.

Vous pouvez partir sur une représentation de graphes avec des listes d'adjacences pour chaque sommet. Vous prévoierez bien sûr les fonctions de base pour accéder aux noeuds et aux arêtes du graphes, et pour créer/modifier le graphe. Prévoyez aussi une méthode pour afficher le contenu du graphe de façon pratique, au moins pour débugger.

2 Graphes pondérés

On veut maintenant gérer le fait que les arêtes ont un poids, représentant des distances entre les sommets. Modifier votre classe pour prévoir ce cas là.

3 Application

Récupérer le fichier stormofswords, qui décrit un graphe pondéré listant des relations entre personnages d'une série de romans bien connue. Plus le poids est élevé, plus deux personnages sont "proches" dans l'histoire.

Utilisez votre classe pour trouver la ou les composantes connexes du graphe, et identifiez les personnages centraux dans chaque composante selon les deux critères suivants:

1. personnage avec le plus de liens
2. personnage le plus "près" des autres personnages en moyenne (en utilisant les plus courts chemins, avec ou sans les poids).

3.0.1 Guide pratique de l'affichage avec format

Comme en C avec printf, on peut utiliser un patron de formatage, en remplaçant les symboles %d, %f, %s (entre autres) par des variables de type entier, float, ou chaines. Par défaut, %s accepte tout type qui peut s'afficher (pour lequel une méthode de représentation est définie).

[2]:
```
a = 2.5
print("%s est tout ce qu'on veut"%a)
```

```
2.5 est tout ce qu'on veut
```

[3]:
```
b = "un flottant?"
print("%s = %s"%(a,b))
```

```
2.5 = un flottant?
```

[4]:
```
print ("%d ~= %f"%(2,2.0001))
```

```
2 ~= 2.000100
```

On peut aussi directement mettre des variables avec une chaine spéciale f"..."

[5]:
```
print(f"{a} = {b} ?")
```

```
2.5 = un flottant? ?
```