

Algorithmes et structures de données avancées : TD 7

Graphes - Matrice d'Adjacence - algorithmes sur les graphes

- Dans ce TP, nous travaillons exclusivement sur des graphes non-orientés.
- Pour connaître la taille d'une liste ou d'une liste des listes (par exemple dans une fonction ...), vous pouvez vous en servir de la fonction python `len`.
- Pour initialiser des listes avec `n` éléments, vous disposez de l'instruction suivante :

```
l = [ 0 for i in range(n) ]
```

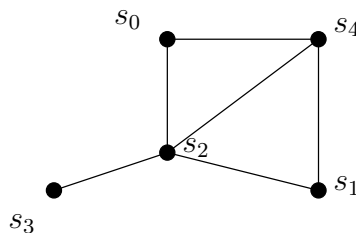
- Pour initialiser une matrice qui est stockée dans une liste des listes `A` avec `lignes` lignes et `cols` colonnes vous disposez de l'instruction suivante :

```
A = [  
    [ 0 for j in range(cols) ]  
    for i in range(lignes)  
]
```

Exercice 7.1 *Matrice d'adjacence pour un graphe non-orienté*

Dans cet exercice, vous allez élaborer pas-à-pas votre première structure de données pour les graphes ainsi que vos premières algorithmes sur les graphes.

1. Déclarer une variable `A` qui remplit la matrice d'adjacence avec le graphe **non-orienté** suivant :



2. Ecrire une fonction `def arete(A, s1, s2):` qui retourne `True` s'il y a une arête entre les sommets `s1` et `s2`, et `False` sinon.
3. Tester votre fonction `def arete(A, s1, s2):` en l'appelant pour quelques combinaisons de sommets.
4. Ecrire une fonction `def voisins(A, sommet):` qui affiche à l'écran les voisins du sommet numéro `sommet` du graphe représenté par la matrice d'adjacence `A` à l'écran (de la même manière que l'exemple suivant du sommet `s1`) :

Sommet s_0 a comme voisin : s_2 s_4

5. Tester votre fonction `def voisins(A, sommet)`: en l'appelant pour chaque sommet du graphe.

Exercice 7.2 *Algorithmes sur des graphes non-orientés*

1. Ecrire une fonction `def degre(A, sommet)`: qui renvoie le degré du sommet numéro `sommet` du graphe représenté par la matrice d'adjacence `A`.
2. Tester votre fonction en l'appelant pour chaque sommet du graphe.
3. Ecrire une fonction `def degreMaximum(A)`: qui renvoie le degré maximum de tous les sommets du graphe représenté par la matrice d'adjacence `A`. Utiliser la fonction `def degre(A, sommet)`: que vous avez implémentée précédemment.
4. Tester votre fonction.
5. Quelle est la complexité asymptotique de la fonction `degre` ?
6. Quelle est la complexité asymptotique de la fonction `degreMaximum` ? Quelle règle avez-vous appliquée ?

Exercice 7.3 *Algorithmes sur des graphes non-orientés*

1. Ecrire une fonction `def nombreAretes(A)`: qui renvoie le nombre d'arêtes du graphe représenté par la matrice d'adjacence `A`.
2. Quelle est la complexité asymptotique de votre fonction `nombreAretes` ?

Exercice 7.4 *Algorithmes sur des graphes non-orientés*

1. Ecrire une fonction `rajouterArete` qui prends en paramètre un graphe représenté par une matrice d'adjacence, ainsi que deux sommets `s` et `t`, et qui permet de rajouter l'arête représenté par les deux sommets `s` et `t` au graphe.
2. Appeler cette fonction et pour qu'elle rajoute une arête entre les sommets s_0 et s_3 .
3. Vérifier que vos changements ont pris effet en appelant la fonction `voisin`.
4. Quelle est la complexité asymptotique de votre fonction `rajouterArete` ?

Exercice 7.5 *Algorithmes sur des graphes non-orientés*

1. Ecrire une fonction `def afficher(A)`: qui permet d'afficher la matrice d'adjacence représenté par la matrice d'adjacence `A`.