TD Informatique TSI2

GRAPHS : Traversal algorithms

EXERCICE 01

On rappelle le code Python pour la fonction `parcours_en_largeur` ci-dessous. On a ajouté par rapport à la version du cours quelques `print` pour visualiser mieux les différentes étapes.

```
In [1]:  def parcours_en_largeur(graphe,sommet):
    ...:      file = [sommet]
    ...:      sommets_visites = []
    ...:      while len(file) != 0 :
    ...:          S = file[0] ; print("sommet a explorer:",S)
    ...:          for voisin in graphe[S]:
    ...:              if not(voisin in sommets_visites) and not(voisin in file):
    ...:                  file.append(voisin)
    ...:                  print("Construction file voisins non visited:",file)
    ...:          sommets_visites.append(file.pop(0))
    ...:          print("Sommets visited:",sommets_visites)
    ...:      return sommets_visites
```

Taper ce code Python puis rentrer dans `graphe` sous la forme d'un dictionnaire le graphe à 8 sommets pris pour exemple dans le cours. On rangera les clés selon l'alphabet. Puis taper `parcours_en_largeur(graphe,'A')`.

EXERCICE 02

On rappelle le code Python pour la fonction `parcours_en_profondeur` ci-dessous. On a ajouté encore par rapport à la version du cours quelques `print` pour visualiser mieux les différentes étapes.

```
In [5]: def parcours_en_profondeur(graphe,sommet):
    ...:      pile = [sommet]
    ...:      sommets_visites = []
    ...:      while len(pile) != 0 :
    ...:          S = pile.pop(); print("sommet a explorer:",S)
    ...:          sommets_visites.append(S)
    ...:          print("sommets visited:",sommets_visites)
    ...:          for voisin in graphe[S]:
    ...:              if not(voisin in sommets_visites) and not(voisin in pile):
    ...:                  pile.append(voisin)
    ...:                  print("construction de la pile des non visited :",pile)
    ...:      return sommets_visites
```

Taper ce code Python puis taper `parcours_en_profondeur(graphe,'A')`, où `graphe` est le graphe à 8 sommets pris pour exemple dans le cours (sous forme d'un dictionnaire).

T.S.V.P →

EXERCICE 03

Parmi les assertions suivantes, lesquelles sont vraies ?

A. On peut utiliser une file pour programmer avec Python un parcours en largeur.

B. Dans un parcours en profondeur, on commence par visiter tous les sommets adjacents au sommet de départ.

C. On peut détecter la présence d'un cycle dans un graphe non orienté avec un parcours en largeur.

D. On ne peut pas détecter la connexité d'un graphe non orienté avec un parcours en profondeur.

E. La complexité d'un parcours en profondeur dans un graphe d'ordre n est linéaire en n.

EXERCICE 04

Recherche d'un cycle dans un graphe connexe non orienté

Soient les graphes connexes non orientés et non pondérés de matrices d'adjacence respectives :

$$M_1 = \begin{pmatrix} 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 \\ 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \end{pmatrix} \text{ et } M_2 = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix}.$$

1. On veut visualiser le graphe correspondants à M_1. Taper le code suivant.

```
In [7]: import networkx as nx

In [8]: import matplotlib.pyplot as plt

In [9]: import numpy as np

In [10]: M1 = np.array([[0,0,1,1,1,0],[0,0,1,1,1,0],[1,1,0,0,0,1],
[1,1,0,0,0,1 ],[1,1,0,0,0,0],[0,0,1,0,0,0]])

In [11]: graphe1=nx.Graph(M1)

In [12]: position1= nx.circular_layout(graphe1)

In [13]: nx.draw(graphe1,with_labels=True,pos=position1)
```

   On verra apparaître un graphe (celui de M_1) dont les sommets par défaut sont numérotés de 0 à 5. On pourrait remplacer par des lettres mais c'est ici sans intérêt et il vaut mieux garder des chiffres qui partent de 0 en vu du programme suivant.

2. De même, visualiser le graphe `graphe2` associé à la matrice M_2.

3. On rappelle qu'un cycle est un chemin dans un graphe tel que toutes les arêtes sont différentes partant et aboutissant au même sommet.
   Warning, dans un graphe non orienté, le chemin $(1 \to 2 \to 1)$ n'est pas un cycle car on a deux fois la même arête (par contre si le graphe est orienté, c'est un cycle).

   À la main, pouvez-vous trouver des cycles dans `graphe1` partant et aboutissant à chaque sommet ? Faire de même avec `graphe2` ?

4. On considère le programme suivant qui permet de détecter si un graphe possède au moins un cycle. La fonction retourne `False` si la matrice `M` ne possède pas de cycle et `True` si elle en a au moins un. Attention, on part d'un sommet du graphe nommé `debut` mais ce sommet n'est pas nécessairement le départ du cycle s'il y en a un.

```
In [49]: def cycle_som(M,debut):
    ...:     from collections import deque
    ...:     n=len(M)
    ...:     FATHER=[-1 for i in range(n)]
    ...:     couleur = ["blanc" for i in range(n)]
    ...:     D = deque()
    ...:     D.append(debut)
    ...:     couleur[debut] = "gris"
    ...:     while len(D) != 0 :
    ...:         x = D.popleft(); print("x vaut :",x)
    ...:         couleur[x] = "noir"
    ...:         for i in range(n):
    ...:             if M[x][i]>0 and couleur[i]!="blanc" and FATHER[x]!=i:
    ...:                 return True
    ...:             elif M[x][i]>0 and couleur[i] == "blanc":
    ...:                 D.append(i)
    ...:                 FATHER[i] = x
    ...:                 couleur[i]="gris"
    ...:     return False
```

   (a) Faire à la main `cycle_som(M1,0)` en explicitant les transformations successives de `FATHER` et de `couleur`. Puis taper la fonction `cycle_som(M,debut)` et appliquer à `M1` et à `0`

   (b) Comprendre alors comment procède la fonction booléenne `cycle_som`
   Taper ensuite `cycle_som(M1,i)` pour tous les sommets `i` de `graphe1` et retrouver le fait qu'en partant dans l'algorithme de chaque sommet, même d'un sommet qui n'appartient pas à un cycle, on puisse créer des cycles.

   (c) On suppose que la ligne `M[0:]` de la matrice d'adjacence ne possède que des `0`.
   Que va afficher `cycle_som(M,0)` ? Conclure.

   (d) De même, faire à la main `cycle_som(M2,0)` puis taper le. Taper ensuite `cycle_som(M1,i)` pour tous les sommets `i` de `graphe2` et retrouver le fait que l'on a aucun cycle dans `graphe2`

   (e) Écrire un programme `research_cycle(M)` qui affiche `True` si `M` possède au moins un cycle et `False` sinon.

   Indication : *on fait une boucle* `for k in range(n):` *où* `k` *est un sommet quelconque du graphe de matrice d'adjacence* `M` *et on testera* `if cycle_som(M,k) == True:`