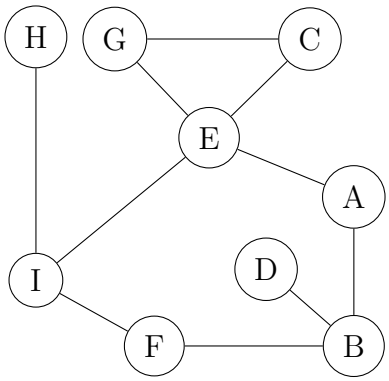
Devoir surveillé n°8 – NSI
23/02/2024

Exercice 1 (6 points)

On considère le graphe ci-contre.



1. Quel est l'ordre du graphe ?
2. Donner l'ordre des sommets visités lors d'un parcours en profondeur partant du sommet A.
3. Donner l'ordre des sommets visités lors d'un parcours en largeur partant du sommet D.
4. Déterminer la matrice d'adjacence du graphe.
5. Déterminer la liste d'adjacence du graphe comme vue en cours.
6. On appelle **densité** d'un graphe non orienté le nombre :

$$D = \frac{2 \times A}{S(S-1)}$$

où A est le nombre d'arêtes et S est le nombre de sommets du graphe.

Un graphe est dit « *creux* » si D est proche de 0. Il est dit « *dense* » si D est proche de 1.

(a) Calculer la densité du graphe.

(b) Le graphe est-il plutôt dense, ou plutôt creux ?

Exercice 2 (11 points)

Dans un graphe non orienté, il peut exister des **composantes connexes**, c'est-à-dire des groupes de sommets accessibles entre eux mais à partir desquels on ne peut pas accéder au reste du graphe. La composante connexe d'un sommet est le groupe de sommets du graphe accessibles entre eux qui contient ce sommet.

1. Dessiner un graphe non orienté d'ordre 6 ayant trois composantes connexes.

   Donner alors les composantes connexes de ce graphe.

On dispose d'une classe `Graphe_no` permettant de définir des objets représentant des graphes non orientés.
Elle dispose entre autres des méthodes suivantes (qui seront les seules utilisables ici) :

- `__init__(self)` : constructeur de la classe, qui crée un graphe vide ;
- `ajoute_sommet(self,s)` : ajoute le sommet `s` ;
- `ajoute_arete(self,s1,s2)` : ajoute une arête entre les sommets `s1` et `s2` ;
- `sommets(self)` : retourne la liste des sommets ;
- `voisins(self,s)` : retourne la liste des voisins du sommet `s`.

On dispose également d'une fonction `parcours_profondeur(G,s)` qui prend en argument un graphe `G` et un sommet `s` et qui retourne la liste des sommets visités lors du parcours en profondeur du graphe `G` en partant du sommet `s`.
On dispose enfin d'une fonction `soustraire(elements,liste)` qui soustrait (supprime) de la liste `liste` les éléments de la liste `elements`.

2. Recopier et compléter la fonction `composantes_connexes(G)` ci-après, qui prend en argument un graphe `G`, et qui retourne la liste des composantes connexes, une composante connexe étant donnée sous la forme de la liste de ses sommets.

   Indication : il faudra utiliser la fonction `parcours_profondeur(G,s)`.

```
def composantes_connexes(G):
    composantes=[] # contiendra les composantes connexes de G
    non_visites = ... # liste des sommets non encore visités de G
    while non_visites !=[]:
        s = non_visites[0]
        composante = ... # composante connexe du sommet s
        soustraire(...,...)
        composantes.append(composante)
    return ...
```

3. On souhaite définir une fonction `sous_graphe(G,composante)` qui, étant donné un graphe `G` et une de ses composantes connexes `composante`, retourne un graphe dont les sommets sont ceux de la composante et dont les arêtes sont les mêmes, pour ces sommets, de ceux de `G`.

   Recopier et compléter son code ci-dessous :

```
def sous_graphe(G,composante):
    g = ... # création d'un graphe g vide
    for s in composante:
        ... # ajouter le sommet s au graphe g
    for s in g.sommets():
        for v in ...:
            ... # ajouter à g une arête entre s et v
    return g
```

4. On souhaite maintenant rendre le graphe complètement connexe.

   Recopier et compléter la fonction `reconnecte(G)` suivante qui ajoute des arêtes à `G` afin de n'avoir plus qu'une composante connexe.

   Indication : L'idée est de relier, pour chaque composante de la liste des composantes, un sommet de celle-ci à un sommet de la composante suivante.

   Le choix de sommet est arbitraire (et laissé libre), mais ne doit pas être effectué au hasard.

```
def reconnecte(G):
    composantes = composantes_connexes(G)
    for i in range(len(composantes)-1):
        s1 = ... # choix d'un sommet de la composante d'indice i
        s2 = ... # choix d'un sommet de la composante suivante
        ... # ajout de l'arête reliant s1 à s2
```

5. Expliquer pourquoi il y a un « `-1` » dans la ligne

   « `for i in range(len(composantes)-1):` » de la fonction précédente.

6. **(question facultative comptée en bonus (+3 points))**

   Modifier la fonction précédente pour que le sommet choisi soit à chaque fois l'un de ceux qui a le plus petit degré de sa composante connexe.

   Pour cela, on pourra définir une fonction intermédiaire `choix_sommet(G,composante)` qui prend en argument un graphe `G` et une composante `composante` (liste de sommets) et qui retourne un des sommets de degré minimal.

   On pourra en plus supposer ici accessible la méthode `degre(self,s)` de la classe `Graphe_no` qui donne le degré d'un sommet `s`.