Cours IV

Parcours de graphes

Lycée Louis Thuillier - Informatique - PCSIB -

Table des matières

Parcourir un graphe revient à créer un algorithme, *i.e.* un liste finie d'instruction, qui permet de visiter les sommets du graphe les uns après les autre.

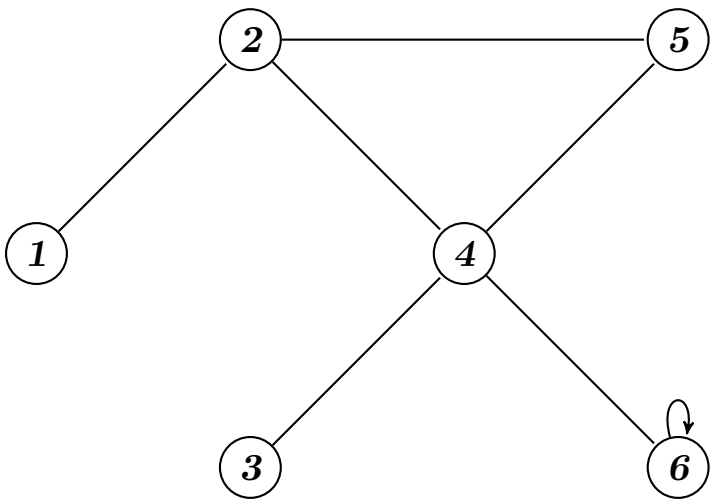
> ***Exemple 1 :*** *On imagine un archipel d'iles et un automate doit se rendre dans chacune des îles pour y livrer des colis. On représente le problème par un graphe dont les noeuds sont les îles et les arêtes les ponts reliant les îles.*
> *On va chercher à donner à l'automate une liste d'instructions pour qu'il puisse se rendre dans toutes les îles et livrer ses colis : on cherche à parcourir le graphe.*

1 Parcours en profondeur

1.1 Algorithme

Pour parcourir un graphe en profondeur, on suit les chemins (arêtes ou arcs) dans le graphe le plus loin possible en marquant les sommets qu'on visite. On est bloqué sur un sommet lorsque plus aucun sommet n'est accessible ou que les sommets accessibles ont déjà été visités.
Lorsque cela arrive, on revient en arrière sur le sommet précédent et on reprend l'exploration. A force d'appliquer cette règle, on revient automatiquement au sommet de départ où on finit bloqué : le parcours est terminé.

> ***Exemple 2 :*** *Parcourir en profondeur le graphe à partir du sommet 1 et 4.*
> *Merci à Paul pour ce graphe très pratique pour ces exemples !*



1.2 Implémentation en Python par récursivité

On souhaite écrire une fonction récursive qui permet de visiter un graphe en profondeur. Un graphe sera représenté par un dictionnaire.

Exemple 3 :

▷ *Comment accéder à partir du dictionnaire représentant un graphe à la liste des sommets voisins d'un sommet s.*

▷ *On suppose qu'on a déjà visité un certains nombre de sommets, enregistrés dans une liste "deja_visites". Compléter la fonction $Continuation\_Parcours\_Profondeur$ qui prend en entrée un sommet s, un graphe $Graphe$ et une liste $deja\_visites$ des sommets déjà visités et qui affiche les sommets visités en continuant le parcours en profondeur à partir du sommet s.*

▷ *En déduire une fonction $Parcours\_Profondeur$ qui prend en entrée un sommet s et un graphe $Graphe$ et qui affiche les sommets visités par un parcours en profondeur partant de s.*

```
def Continuation_Parcours_Profondeur (s, graphe, deja_visites):

    deja_visites.append( .... )

    print(s)

    for sommet_voisin in ....... :

        if sommet_voisin not in ..............  :

            ..............................................
```

1.3 Notion de pile et parcours en profondeur

▶ Une pile

En programmation une pile est une structure de données (typiquement une liste) à laquelle on va
▷ ajouter des éléments un par un par la droite : c'est l'**empilage**
▷ enlever des éléments un à un par la droite : c'est le **dépilage**

En Python, avec une liste :
▷ on empile avec *append*()
▷ on dépile avec *pop*()
Attention ! *pop* enlève l'élément de la liste **ET** renvoie cet élément.

```
>>> l=[1,2,3,4]

>>> x=l.pop()

>>> print(x)
4

>>> print(l)
[1, 2, 3]
```

▶ Utilisation d'une pile pour un parcours en profondeur

Pour parcourir un graphe en profondeur, on va créer en parallèle
▷ une pile des sommets dont il faut encore explorer les voisins
▷ une liste des sommets déjà explorés

Attention ! Dès qu'on commence à explorer les voisins d'un sommet, on l'enlève de la pile.

Astuce : la pile peut être vu comme "la pile des trucs à faire", ici la pile des sommets dont il va falloir visiter le voisin.

▶ **Implémentation en Python**

```
def parcours_profondeur_pile (s, graphe):
    deja_visites = .......
    pile_a_visiter=   [ s ]
    while len(pile_a_visiter)........ :
        sommet= ....................
        if sommet not in  ...........  :
            deja_visites.append( .............. )
            print( .......... )

            for sommet_voisin in  ............. :

                pile_a_visiter.append( ......... )
```

2 Parcours en largeur

2.1 Algorithme

Pour un parcours en largeur, on va visiter les sommets du graphe en ronds concentriques. On part du sommet de départ

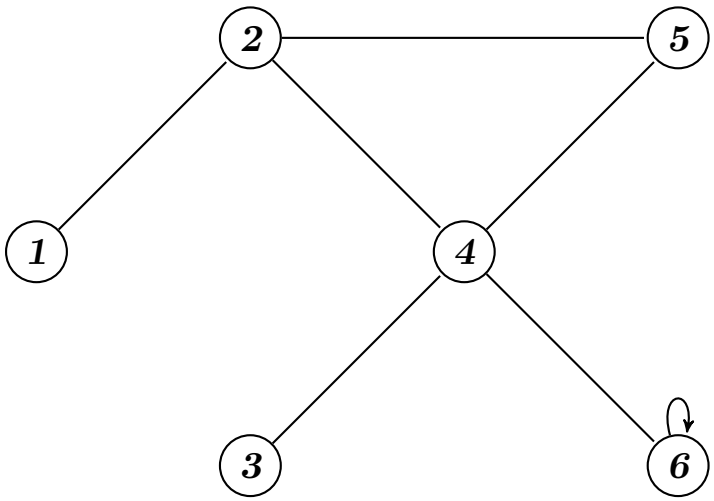
- ▷ on visite tous les voisins du sommet de départ
- ▷ puis on visite les voisins des voisins du sommet de départ
- ▷ et ainsi de suite

A chaque fois, on note évidemment les sommets visités et on ne les "revisite" pas.

Différence entre un parcours en profondeur et en largeur

- ▷ Pour un parcours en profondeur, on suit un chemin jusqu'à arrivé à un cul de sac.
- ▷ Pour un parcours en largeur, on cherche par zone : tous les sommets reliés au sommet initial par un chemin de longueur 1, puis tous les sommets reliés pas un chemin de longueur 2, ...

Exemple 4 : Parcourir en largeur le graphe à partir du sommet 1 et 4.



2.2 Notion de file et parcours en largeur

▶ Une file

En programmation une file est une structure de données à laquelle on va
▷ ajouter des éléments un par un par la droite : c'est l'**enfilage**
▷ enlever des éléments un à un par la gauche : c'est le **défilage**
Les listes ne sont pas des objet bien adaptés aux files. Par exemple l'ajout et la suppression d'éléments se font tous les deux par la droite.
On utilisera par la suite le type *deque* ("double-entry queue" en anglais) du module *collections*.

```
>>> from collections import deque

>>> file=deque([1])

>>> print(file)
deque([1])
```

Avec un tel objet on peut :
▷ ajouter un élément par la droite via la commande *append*()
▷ supprimer un élément par la gauche avec la commande $popleft()$

```
>>> file.append(2)

>>> file.append(3)

>>> file.popleft()
1

>>> print(file)
deque([2, 3])
```

Attention ! Différence entre une pile et une liste
Ce sont deux façon d'organiser des tâches à accomplir. La différence entre une pile et une file est comme dans la vraie vie :
▷ lorsqu'on empile des dossiers à traiter, le dossier que l'on traite en premier est le dernier posé sur la pile.
⇒ Une pile est une structure de données de type **LIFO** : "Last In First Out".
▷ au contraire dans une file d'attente, la personne prise en charge par le guichet est la première arrivée des personnes qui attendent.
⇒ Une file est une structure de données de type **FIFO** : "First In First Out".

▶ Utilisation d'une file pour un parcours en largeur

Pour programmer à l'aide d'une file le parcours d'un graphe en largeur, on va créer une file des sommets qu'on visite et on va sauvegarder dans une liste les sommets déjà visités.

▷ On met le sommet de départ dans la file.
▷ On visite ses voisins (donc on les ajoute à la liste des sommets visités), on ajoute ces sommets à la file (car on va visiter leurs voisins), puis on enlève le sommet initial de la pile (on a visiter tous ses voisins, c'est bon pour lui).

▷ On recommence!!
On choisit le premier sommet de la file, on visite ses voisins qui n'ont pas encore été visités, on les mets dans la file puis on enlève le premier sommet de la file.
Et ainsi de suite ...
▷ On s'arrête lorsque la file est vide.

Exemple 5 :

```
def parcours_largeur_file(s, graphe):

    deja_visites = [s]
    file_visite=deque([s])
    while len(file_visite)!=0 :

        sommet=.............

        print(sommet)

        for sommet_voisin in ........... :

            if ..................... :

                ..........................

                ..............................
```