Informatique tronc commun - PCSI

Cours 6 :
Parcours de graphe

Florent Pompigne
pompigne@crans.org

Lycée Buffon

année 2021/2022

Principe

Dans la suite, on fixe un sommet s (pour source) du graphe.
On dit qu'un sommet u est **accessible** depuis s s'il existe un chemin de s à u.

Principe

Dans la suite, on fixe un sommet s (pour source) du graphe. On dit qu'un sommet u est **accessible** depuis s s'il existe un chemin de s à u.

Un **parcours de graphe** est un algorithme qui visite une et une seule fois chaque sommet accessible depuis s. L'opération à effectuer sur chaque sommet au moment où on le visite dépendra du contexte dans lequel on utilise ce parcours de graphe.

Principe

Dans la suite, on fixe un sommet s (pour source) du graphe. On dit qu'un sommet u est **accessible** depuis s s'il existe un chemin de s à u.

Un **parcours de graphe** est un algorithme qui visite une et une seule fois chaque sommet accessible depuis s. L'opération à effectuer sur chaque sommet au moment où on le visite dépendra du contexte dans lequel on utilise ce parcours de graphe.

On distingue deux types de parcours classiques :

Principe

Dans la suite, on fixe un sommet s (pour source) du graphe. On dit qu'un sommet u est **accessible** depuis s s'il existe un chemin de s à u.

Un **parcours de graphe** est un algorithme qui visite une et une seule fois chaque sommet accessible depuis s. L'opération à effectuer sur chaque sommet au moment où on le visite dépendra du contexte dans lequel on utilise ce parcours de graphe.

On distingue deux types de parcours classiques :

- le **parcours en largeur**, où on explore tous les sommets à distance k de s avant de passer à ceux de distance $k + 1$;

Principe

Dans la suite, on fixe un sommet s (pour source) du graphe. On dit qu'un sommet u est **accessible** depuis s s'il existe un chemin de s à u.

Un **parcours de graphe** est un algorithme qui visite une et une seule fois chaque sommet accessible depuis s. L'opération à effectuer sur chaque sommet au moment où on le visite dépendra du contexte dans lequel on utilise ce parcours de graphe.

On distingue deux types de parcours classiques :

- le **parcours en largeur**, où on explore tous les sommets à distance k de s avant de passer à ceux de distance $k+1$;
- le **parcours en profondeur**, où on visite à chaque étape un successeur du dernier sommet visité qui en a un.

Principe

Dans la suite, on fixe un sommet s (pour source) du graphe. On dit qu'un sommet u est **accessible** depuis s s'il existe un chemin de s à u.

Un **parcours de graphe** est un algorithme qui visite une et une seule fois chaque sommet accessible depuis s. L'opération à effectuer sur chaque sommet au moment où on le visite dépendra du contexte dans lequel on utilise ce parcours de graphe.

On distingue deux types de parcours classiques :

- le **parcours en largeur**, où on explore tous les sommets à distance k de s avant de passer à ceux de distance $k+1$;
- le **parcours en profondeur**, où on visite à chaque étape un successeur du dernier sommet visité qui en a un.

On décrit dans la suite ces parcours pour des graphes représentés par listes d'adjacences. On notera n le nombre de sommets du graphe et m le nombre d'arêtes.

Parcours en largeur

Le parcours en largeur utilise une **file** dans laquelle chaque sommet va être placé au moment où il est visité, puis sorti au moment où le parcours continue sur ses voisins. Une file est une structure suivant l'ordre FIFO : First In First Out, les sommets en sortent donc dans l'ordre où ils y sont entrés. Les sommets seront donc visités dans l'ordre de leur distance à s : d'abord s, puis les voisins de s, puis les voisins des voisins, etc...

Parcours en largeur

Le parcours en largeur utilise une **file** dans laquelle chaque sommet va être placé au moment où il est visité, puis sorti au moment où le parcours continue sur ses voisins. Une file est une structure suivant l'ordre FIFO : First In First Out, les sommets en sortent donc dans l'ordre où ils y sont entrés. Les sommets seront donc visités dans l'ordre de leur distance à s : d'abord s, puis les voisins de s, puis les voisins des voisins, etc...

On dira qu'un sommet visité est **marqué** pour s'assurer qu'il n'est pas visité une seconde fois.

Pseudo-code du parcours en largeur

```
parcours_largeur(G,s)
    les sommets sont initialement non marqués
    on marque s
    on crée une file vide F
    on enfile s dans F
    # s est visité
    tant que F est non vide
        on défile un sommet u de F
        Pour chaque v successeur non marqué de u
            on marque v
            on enfile v dans F
            # v est visité
```

Pseudo-code du parcours en largeur

```
parcours_largeur(G,s)
    les sommets sont initialement non marqués
    on marque s
    on crée une file vide F
    on enfile s dans F
    # s est visité
    tant que F est non vide
        on défile un sommet u de F
        Pour chaque v successeur non marqué de u
            on marque v
            on enfile v dans F
            # v est visité
```

Complexité : Au pire, chaque sommet est visité une fois, avec à chaque fois un marquage, un enfilement et un défilement en $O(1)$, d'où un premier coût en $O(n)$, et par ailleurs toutes les listes d'adjacences sont parcourues, d'où un second coût en $O(m)$. La complexité totale est donc en $O(n+m)$.

Parcours en profondeur

Le parcours en profondeur utilise une **pile** dans laquelle chaque sommet va être ajouté au moment où il est visité, puis sorti au moment où le parcours continue sur un de ses voisins. Une pile est une structure suivant l'ordre LIFO : Last In First Out, l'élément au **sommet** de la pile est donc le dernier élément ajouté.

Parcours en profondeur

Le parcours en profondeur utilise une **pile** dans laquelle chaque sommet va être ajouté au moment où il est visité, puis sorti au moment où le parcours continue sur un de ses voisins. Une pile est une structure suivant l'ordre LIFO : Last In First Out, l'élément au **sommet** de la pile est donc le dernier élément ajouté.

L'idée du parcours en profondeur est donc d'avancer autant que possible dans la partie non marquée du graphe, et de ne revenir en arrière que lorsqu'on se trouve dans un cul-de-sac.

Pseudo-code du parcours en profondeur

```
parcours_profondeur(G,s)
    les sommets sont initialement non marqués
    on crée une pile vide P
    on empile s dans P
    tant que P est non vide
        on dépile un sommet u de P
        si u est non marqué
            on marque u
            # u est visité
            Pour chaque v successeur non marqué d
                on empile v
```

Pseudo-code du parcours en profondeur

```
parcours_profondeur(G,s)
    les sommets sont initialement non marqués
    on crée une pile vide P
    on empile s dans P
    tant que P est non vide
        on dépile un sommet u de P
        si u est non marqué
            on marque u
            # u est visité
            Pour chaque v successeur non marqué d
                on empile v
```

Complexité : Toujours en $O(n+m)$.

Parcours en profondeur en récursif

Il est également possible d'écrire un parcours en profondeur récursif (le rôle de la pile précédente est alors joué par la pile d'appel) :

```
parcours_profondeur(G,s)
    visiter(u)
        on marque u
        # u est visité
        Pour chaque v successeur de u
            Si v est non marqué
                visiter(v)
    les sommets sont initialement non marqués
    visiter(s)
```

Parcours en profondeur en récursif

Il est également possible d'écrire un parcours en profondeur récursif (le rôle de la pile précédente est alors joué par la pile d'appel) :

```
parcours_profondeur(G,s)
    visiter(u)
        on marque u
        # u est visité
        Pour chaque v successeur de u
            Si v est non marqué
                visiter(v)
    les sommets sont initialement non marqués
    visiter(s)
```

Complexité : toujours en $O(n+m)$.

Implémentation en Python

Le détail de l'implémentation sera étudié en TP. Notons les points suivants :

- on marquera les sommets à l'aide d'un tableau (ou d'un dictionnaire) de booléens ;

Implémentation en Python

Le détail de l'implémentation sera étudié en TP. Notons les points suivants :

- on marquera les sommets à l'aide d'un tableau (ou d'un dictionnaire) de booléens ;
- pour parcourir les successeurs d'un sommet u, il suffit de parcourir la liste d'adjacence de u.

Implémentation en Python

Le détail de l'implémentation sera étudié en TP. Notons les points suivants :

- on marquera les sommets à l'aide d'un tableau (ou d'un dictionnaire) de booléens ;
- pour parcourir les successeurs d'un sommet u, il suffit de parcourir la liste d'adjacence de u.
- une pile peut être représenté en Python par une liste, le sommet étant l'élément en fin de liste. On peut alors empiler avec `append` et dépiler avec `pop` ;

Implémentation en Python

Le détail de l'implémentation sera étudié en TP. Notons les points suivants :

- on marquera les sommets à l'aide d'un tableau (ou d'un dictionnaire) de booléens ;
- pour parcourir les successeurs d'un sommet u, il suffit de parcourir la liste d'adjacence de u.
- une pile peut être représenté en Python par une liste, le sommet étant l'élément en fin de liste. On peut alors empiler avec `append` et dépiler avec `pop` ;
- en revanche, pour représenter une file par une liste, il faudrait défiler en début de liste. On peut le faire avec `pop(0)`, mais ce n'est pas en $O(1)$. Nous utiliserons les files à double entrée `deque` de la bibliothèque `collections`, qui disposent d'une opération `popleft` en $O(1)$.

Application à la connexité

Par principe, à l'issue d'un parcours, tous les sommets accessibles depuis *s* ont été visités et sont donc marqués. Un graphe non orienté est donc connexe si et seulement si tous les sommets sont marqués après le parcours. Si le graphe n'est pas connexe, les sommets marqués forment la **composante connexe** de *s*.

Calcul de la distance par un parcours en largeur

Un parcours en largeur permet de calculer pour chaque sommet accessible u la distance $d(u)$ de s à u et le plus court chemin pour aller de s à u, représenté en notant le prédécesseur `pred(u)` de u.

Calcul de la distance par un parcours en largeur

Un parcours en largeur permet de calculer pour chaque sommet accessible u la distance $d(u)$ de s à u et le plus court chemin pour aller de s à u, représenté en notant le prédécesseur `pred(u)` de u.

```
parcours_largeur(G,s)
    les sommets sont initialement non marqués
    on marque s
    on crée une file vide F
    on enfile s dans F
    on pose d(s) = 0
    tant que F est non vide
        on défile un sommet u de F
        Pour chaque v successeur non marqué de u
            on marque v
            on enfile v dans F
            on calcule d(v) = d(u) + 1
            on note pred(v) = u
```