

Parcours de graphes

Prof d'info

Lycée Thiers

1 Parcours en largeur

2 Parcours en profondeur

- Un *parcours* permet d'explorer les sommets d'un graphe de proche en proche à partir d'un sommet initial.

Présentation

- Un *parcours* permet d'explorer les sommets d'un graphe de proche en proche à partir d'un sommet initial.
- Deux parcours classiques :

- Un *parcours* permet d'explorer les sommets d'un graphe de proche en proche à partir d'un sommet initial.
- Deux parcours classiques :
 - le parcours en *largeur* (BFS, *Breadth-First Search*);

- Un *parcours* permet d'explorer les sommets d'un graphe de proche en proche à partir d'un sommet initial.
- Deux parcours classiques :
 - le parcours en *largeur* (BFS, *Breadth-First Search*) ;
 - le parcours en *profondeur* (DFS, *Depth-First Search*).

- Un *parcours* permet d'explorer les sommets d'un graphe de proche en proche à partir d'un sommet initial.
- Deux parcours classiques :
 - le parcours en *largeur* (BFS, *Breadth-First Search*) ;
 - le parcours en *profondeur* (DFS, *Depth-First Search*).
- Ils servent notamment à étudier l'accessibilité et la connexité.

Deux couleurs

- Pour ne pas rencontrer plusieurs fois le même sommet, on le marque.

Deux couleurs

- Pour ne pas rencontrer plusieurs fois le même sommet, on le marque.
- On utilise seulement deux états :

Deux couleurs

- Pour ne pas rencontrer plusieurs fois le même sommet, on le marque.
- On utilise seulement deux états :
 - **blanc** : le sommet n'a jamais été rencontré ;

Deux couleurs

- Pour ne pas rencontrer plusieurs fois le même sommet, on le marque.
- On utilise seulement deux états :
 - **blanc** : le sommet n'a jamais été rencontré ;
 - **noir** : le sommet a déjà été rencontré.

Deux couleurs

- Pour ne pas rencontrer plusieurs fois le même sommet, on le marque.
- On utilise seulement deux états :
 - **blanc** : le sommet n'a jamais été rencontré ;
 - **noir** : le sommet a déjà été rencontré.
- En Python, un tableau de booléens suffit :

False $\leftrightarrow$ blanc, True $\leftrightarrow$ noir.

Deux couleurs

- Pour ne pas rencontrer plusieurs fois le même sommet, on le marque.
- On utilise seulement deux états :
 - **blanc** : le sommet n'a jamais été rencontré ;
 - **noir** : le sommet a déjà été rencontré.
- En Python, un tableau de booléens suffit :

False $\leftrightarrow$ blanc, True $\leftrightarrow$ noir.

- Un sommet devient noir dès sa première découverte.

1 Parcours en largeur

2 Parcours en profondeur

Rappels : files en Python

```
1 from collections import deque
2 file = deque() # créer une file vide
3 file.append(5) # ajouter 5 à la file
4 file.append(12) # ajouter 12 à la file
5 file.popleft() # retirer le plus ancien élément,
   donc 5
6 print(file) # affiche deque([12])
```

Parcours en largeur

- On part d'un sommet source s .

Parcours en largeur

- On part d'un sommet source s .
- On visite d'abord ses voisins, puis les voisins de ses voisins, et ainsi de suite.

Parcours en largeur

- On part d'un sommet source s .
- On visite d'abord ses voisins, puis les voisins de ses voisins, et ainsi de suite.
- Les sommets découverts sont conservés dans une **file** : premier entré, premier sorti.

Parcours en largeur

- On part d'un sommet source s .
- On visite d'abord ses voisins, puis les voisins de ses voisins, et ainsi de suite.
- Les sommets découverts sont conservés dans une **file** : premier entré, premier sorti.
- Dans un graphe non pondéré, le BFS visite donc les sommets par distance croissante depuis s .

Parcours en largeur : Python

On suppose le graphe représenté par un tableau de listes de voisins.

```
1 from collections import deque
2
3 def bfs(g, s):
4     noir = [False] * len(g)
5     file = deque([s])
6     noir[s] = True
7
8     while file:
9         x = file.popleft()
10        print(x)
11        for y in g[x]:
12            if not noir[y]:
13                noir[y] = True
14                file.append(y)
```

Pourquoi noircir dès l'enfilement ?

- Dès qu'un sommet blanc y est découvert, il est ajouté à la file.

Pourquoi noircir dès l'enfilement ?

- Dès qu'un sommet blanc y est découvert, il est ajouté à la file.
- On le noircit **immédiatement**.

Pourquoi noircir dès l'enfilement ?

- Dès qu'un sommet blanc y est découvert, il est ajouté à la file.
- On le noircit **immédiatement**.
- Ainsi, un autre voisin ne pourra pas ajouter une seconde fois y à la file.

Pourquoi noircir dès l'enfilement ?

- Dès qu'un sommet blanc y est découvert, il est ajouté à la file.
- On le noircit **immédiatement**.
- Ainsi, un autre voisin ne pourra pas ajouter une seconde fois y à la file.
- Chaque sommet entre donc au plus une fois dans la file.

1 Parcours en largeur

2 Parcours en profondeur

Parcours en profondeur

- On part d'un sommet x et on poursuit l'exploration par un de ses voisins non encore rencontrés.

Parcours en profondeur

- On part d'un sommet x et on poursuit l'exploration par un de ses voisins non encore rencontrés.
- On continue ainsi le plus loin possible avant de revenir en arrière.

Parcours en profondeur

- On part d'un sommet x et on poursuit l'exploration par un de ses voisins non encore rencontrés.
- On continue ainsi le plus loin possible avant de revenir en arrière.
- La récursivité fournit naturellement la pile nécessaire au parcours.

Parcours en profondeur : Python

On utilise ici une fonction récursive.

```
1 def dfs(g, s):
2     noir = [False] * len(g)
3
4     def explorer(x):
5         noir[x] = True
6         print(x)
7         for y in g[x]:
8             if not noir[y]:
9                 explorer(y)
10
11     explorer(s)
```

Parcourir tout le graphe

Un parcours lancé depuis s ne visite que les sommets accessibles depuis s . Pour parcourir tout le graphe, on relance une exploration depuis chaque sommet encore blanc.

```
1 for s in range(len(g)):  
2     if not noir[s]:  
3         explorer(s)
```

Dans un graphe non orienté, chaque relance correspond à une nouvelle composante connexe.

Complexité (cas d'un graphe orienté)

Soit n le nombre de sommets et p le nombre d'arêtes ou d'arcs.

- Chaque sommet est découvert une fois au plus (et pas 2 à cause de la coloration). Complexité proportionnelle à :

$$\sum_{x=0, \dots, n-1} \left(\underbrace{1}_{\text{découverte}} + \text{traitement des voisins} \right)$$

Complexité (cas d'un graphe orienté)

Soit n le nombre de sommets et p le nombre d'arêtes ou d'arcs.

- Chaque sommet est découvert une fois au plus (et pas 2 à cause de la coloration). Complexité proportionnelle à :

$$\sum_{x=0, \dots, n-1} \left(\underbrace{1}_{\text{découverte}} + \text{traitement des voisins} \right)$$

- Chaque liste de voisins est parcourue une fois. Pour chaque sommet x , le coût est proportionnel au degré c.a.d. au nombre d'arcs sortants de x .

Complexité (cas d'un graphe orienté)

Soit n le nombre de sommets et p le nombre d'arêtes ou d'arcs.

- Chaque sommet est découvert une fois au plus (et pas 2 à cause de la coloration). Complexité proportionnelle à :

$$\sum_{x=0, \dots, n-1} \left(\underbrace{1}_{\text{découverte}} + \text{traitement des voisins} \right)$$

- Chaque liste de voisins est parcourue une fois. Pour chaque sommet x , le coût est proportionnel au degré c.a.d. au nombre d'arcs sortants de x .
- La somme des degrés des sommets est égale au nombre total d'arcs.

Complexité (cas d'un graphe orienté)

Soit n le nombre de sommets et p le nombre d'arêtes ou d'arcs.

- Chaque sommet est découvert une fois au plus (et pas 2 à cause de la coloration). Complexité proportionnelle à :

$$\sum_{x=0, \dots, n-1} \left(\underbrace{1}_{\text{découverte}} + \text{traitement des voisins} \right)$$

- Chaque liste de voisins est parcourue une fois. Pour chaque sommet x , le coût est proportionnel au degré c.a.d. au nombre d'arcs sortants de x .
- La somme des degrés des sommets est égale au nombre total d'arcs.
- Avec une représentation par listes d'adjacence, BFS et DFS ont donc une complexité proportionnelle à :

$$\sum_{x=0, \dots, n-1} (1 + \deg(x)) = n + \sum_{x=0, \dots, n-1} \deg(x) = n + p$$

Coût en $O(n + p)$.

Remarque

Dans le cas d'un graphe non orienté, la somme des degrés est égale à $2p$ car chaque sommet est compté deux fois (une fois comme origine et une autre fois comme extrémité d'arête).

À retenir

	Parcours en largeur	Parcours en profondeur
Structure	file	pile implicite
Idée	explorer par couches	aller le plus loin possible
Complexité	$O(n + p)$	$O(n + p)$