Parcours de graphes

Prof d'info

Lycée Thiers

Crédits

- Sur le BFS

Crédits

- Sur le BFS
- Sur le DFS

Crédits

- Sur le BFS
- Sur le DFS
- A propos des piles et des files

Présentation

- En théorie des graphes, un parcours de graphe est un algorithme consistant à explorer les sommets d'un graphe de proche en proche à partir d'un sommet initial. Un cas particulier important est le parcours d'arbre.

Présentation

- En théorie des graphes, un parcours de graphe est un algorithme consistant à explorer les sommets d'un graphe de proche en proche à partir d'un sommet initial. Un cas particulier important est le parcours d'arbre.
- Deux parcours au programme : le *parcours en profondeur* et le *parcours en largeur*

Présentation

- En théorie des graphes, un parcours de graphe est un algorithme consistant à explorer les sommets d'un graphe de proche en proche à partir d'un sommet initial. Un cas particulier important est le parcours d'arbre.
- Deux parcours au programme : le *parcours en profondeur* et le *parcours en largeur*
- Dans les graphes pondérés : algorithme de *Dijkstra*

Principe

- Un parcours d'un graphe est un procédé qui permet de choisir, à partir des sommets visités, le sommet suivant à visiter.

Principe

- Un parcours d'un graphe est un procédé qui permet de choisir, à partir des sommets visités, le sommet suivant à visiter.
- Le problème consiste à déterminer un ordre sur les visites des sommets.

Principe

- Un parcours d'un graphe est un procédé qui permet de choisir, à partir des sommets visités, le sommet suivant à visiter.
- Le problème consiste à déterminer un ordre sur les visites des sommets.
- Une fois le choix fait, l'ordre des visites induit

Principe

- Un parcours d'un graphe est un procédé qui permet de choisir, à partir des sommets visités, le sommet suivant à visiter.
- Le problème consiste à déterminer un ordre sur les visites des sommets.
- Une fois le choix fait, l'ordre des visites induit
  - une numérotation des sommets visités (l'ordre de leur découverte)

Principe

- Un parcours d'un graphe est un procédé qui permet de choisir, à partir des sommets visités, le sommet suivant à visiter.
- Le problème consiste à déterminer un ordre sur les visites des sommets.
- Une fois le choix fait, l'ordre des visites induit
  - une numérotation des sommets visités (l'ordre de leur découverte)
  - et un choix sur l'arc ou l'arête utilisé pour atteindre un nouveau sommet à partir des sommets déjà visités.

Finalité

Les algorithmes de parcours ne sont pas une fin en eux-mêmes. Ils servent comme outil pour étudier une propriété globale du graphe, par exemple :

- Connexité et forte connexité

Finalité

Les algorithmes de parcours ne sont pas une fin en eux-mêmes. Ils servent comme outil pour étudier une propriété globale du graphe, par exemple :

- Connexité et forte connexité
- Existence d'un circuit ou d'un cycle

Finalité

Les algorithmes de parcours ne sont pas une fin en eux-mêmes. Ils servent comme outil pour étudier une propriété globale du graphe, par exemple :

- Connexité et forte connexité
- Existence d'un circuit ou d'un cycle
- Calcul des plus courts chemins (notamment l'algorithme de Dijkstra)

Finalité

Les algorithmes de parcours ne sont pas une fin en eux-mêmes. Ils servent comme outil pour étudier une propriété globale du graphe, par exemple :

- Connexité et forte connexité
- Existence d'un circuit ou d'un cycle
- Calcul des plus courts chemins (notamment l'algorithme de Dijkstra)
- Calcul d'un arbre recouvrant (notamment l'algorithme de Prim)

Finalité

Les algorithmes de parcours ne sont pas une fin en eux-mêmes. Ils servent comme outil pour étudier une propriété globale du graphe, par exemple :

- Connexité et forte connexité
- Existence d'un circuit ou d'un cycle
- Calcul des plus courts chemins (notamment l'algorithme de Dijkstra)
- Calcul d'un arbre recouvrant (notamment l'algorithme de Prim)
- Algorithmes pour les flots maximums (comme l'algorithme de Ford-Fulkerson).

3 couleurs

- La difficulté de l'exploration consiste à éviter de visiter plusieurs fois un même sommet. Pour cela on met en oeuvre un marquage des sommets par des couleurs.

3 couleurs

- La difficulté de l'exploration consiste à éviter de visiter plusieurs fois un même sommet. Pour cela on met en oeuvre un marquage des sommets par des couleurs.
- Lors d'une exploration, chaque sommet passe par trois couleurs :

3 couleurs

- La difficulté de l'exploration consiste à éviter de visiter plusieurs fois un même sommet. Pour cela on met en oeuvre un marquage des sommets par des couleurs.
- Lors d'une exploration, chaque sommet passe par trois couleurs :
  - bleu tant que la visite du sommet n'a pas commencée

3 couleurs

- La difficulté de l'exploration consiste à éviter de visiter plusieurs fois un même sommet. Pour cela on met en oeuvre un marquage des sommets par des couleurs.
- Lors d'une exploration, chaque sommet passe par trois couleurs :
  - bleu tant que la visite du sommet n'a pas commencée
  - vert dès que sa visite commence et tant le traitement n'est pas terminé

3 couleurs

- La difficulté de l'exploration consiste à éviter de visiter plusieurs fois un même sommet. Pour cela on met en oeuvre un marquage des sommets par des couleurs.
- Lors d'une exploration, chaque sommet passe par trois couleurs :
  - bleu tant que la visite du sommet n'a pas commencée
  - vert dès que sa visite commence et tant le traitement n'est pas terminé
  - rouge dès que le traitement est terminé

3 couleurs

- La difficulté de l'exploration consiste à éviter de visiter plusieurs fois un même sommet. Pour cela on met en oeuvre un marquage des sommets par des couleurs.
- Lors d'une exploration, chaque sommet passe par trois couleurs :
  - bleu tant que la visite du sommet n'a pas commencée
  - vert dès que sa visite commence et tant le traitement n'est pas terminé
  - rouge dès que le traitement est terminé
- L'exploration à partir d'un sommet s ne permet pas nécessairement d'explorer tout le graphe (il peut y avoir plusieurs composantes connexes). Pour effectuer une exploration complète il faut relancer le parcours à partir d'un sommet bleu tant qu'il en existe.

Tableau de couleurs

- Avant l'exploration, on colorie tous les sommets en bleu. En général on utilise un tableau de couleurs (coût en $O(n)$). On a accès au statut d'un sommet (Rouge, Vert, Bleu) en $O(1)$.

Tableau de couleurs

- Avant l'exploration, on colorie tous les sommets en bleu. En général on utilise un tableau de couleurs (coût en $O(n)$). On a accès au statut d'un sommet (Rouge, Vert, Bleu) en $O(1)$.
- Ensuite, on lance l'exploration à partir d'un sommet quelconque.

Tableau de couleurs

- Avant l'exploration, on colorie tous les sommets en bleu. En général on utilise un tableau de couleurs (coût en $O(n)$). On a accès au statut d'un sommet (Rouge, Vert, Bleu) en $O(1)$.
- Ensuite, on lance l'exploration à partir d'un sommet quelconque.
- On gère un accumulateur des sommets verts (suivant les cas : une pile, une file ou autre).

Tableau de couleurs

- Avant l'exploration, on colorie tous les sommets en bleu. En général on utilise un tableau de couleurs (coût en $O(n)$). On a accès au statut d'un sommet (Rouge, Vert, Bleu) en $O(1)$.
- Ensuite, on lance l'exploration à partir d'un sommet quelconque.
- On gère un accumulateur des sommets verts (suivant les cas : une pile, une file ou autre).
- Lors d'un parcours, chaque sommet entre au plus une fois dans l'accumulateur `Verts`, et n'en sort qu'au plus une fois (quand il devient rouge).

Tableau de couleurs

- Avant l'exploration, on colorie tous les sommets en bleu. En général on utilise un tableau de couleurs (coût en $O(n)$). On a accès au statut d'un sommet (Rouge, Vert, Bleu) en $O(1)$.
- Ensuite, on lance l'exploration à partir d'un sommet quelconque.
- On gère un accumulateur des sommets verts (suivant les cas : une pile, une file ou autre).
- Lors d'un parcours, chaque sommet entre au plus une fois dans l'accumulateur `Verts`, et n'en sort qu'au plus une fois (quand il devient rouge).
- On souhaite que ces opérations d'entrée et de sortie dans l'accumulateur soient de coût constant.

Présentation

- Algorithme de parcours en largeur (ou BFS, pour Breadth-First Search en anglais) :

Présentation

- Algorithme de parcours en largeur (ou BFS, pour Breadth-First Search en anglais) :
  - commencer par explorer un nœud source,

Présentation

- Algorithme de parcours en largeur (ou BFS, pour Breadth-First Search en anglais) :
  - commencer par explorer un nœud source,
  - puis ses voisins,

Présentation

- Algorithme de parcours en largeur (ou BFS, pour Breadth-First Search en anglais) :
  - commencer par explorer un nœud source,
  - puis ses voisins,
  - puis les voisins non explorés des voisins, etc.

Présentation

- Algorithme de parcours en largeur (ou BFS, pour Breadth-First Search en anglais) :
  - commencer par explorer un nœud source,
  - puis ses voisins,
  - puis les voisins non explorés des voisins, etc.
- Il permet de déterminer les distances de tous les nœuds depuis un nœud source dans un graphe non pondéré (orienté ou non orienté).

Présentation

- Algorithme de parcours en largeur (ou BFS, pour Breadth-First Search en anglais) :
  - commencer par explorer un nœud source,
  - puis ses voisins,
  - puis les voisins non explorés des voisins, etc.
- Il permet de déterminer les distances de tous les nœuds depuis un nœud source dans un graphe non pondéré (orienté ou non orienté).
- Il peut aussi servir à déterminer si un graphe non orienté est connexe.

Algorithme

- L'ensemble des sommets Verts est représenté par une file

```
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
6 print(file)# affiche deque([12])
```

Algorithme

- L'ensemble des sommets Verts est représenté par une file

```
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
6 print(file)# affiche deque([12])
```

- Principe : on explore le graphe à partir d'un sommet en visitant d'abord tous les sommets voisins (à une distance 1), puis tous les sommets voisins de ses voisins (à une distance 2)....

Algorithme

F : file des sommets verts.

```
1  procedure Largeur(G:graphe, s:sommet)
2    Colorier s en vert et Enfiler s
3    tant que F non vide faire
4      Defiler x
5      pour chaque sommet adjacent y de x :
6        si y est bleu alors
7          Enfiler y /*c.a.d. Colorier y en vert*/
8      Colorier x en rouge
9
10 procedure Largeur_totale (G:graphe)
11   Pour chaque sommet s :
12     si s est bleu alors
13       Largeur(G,s)
14
15 Colorier tous les sommets en bleu
16 Créer file_fifo vide F /*file des sommets verts*/
17 Largeur_totale(G)
```

Complexité

Grapge $G = (S, A)$, $n = |S|$: $p = |A|$

- Les seules opérations effectuées sont les opérations de coloriage, d'enfilement/défilement.

Complexité

Grapge $G = (S, A)$, $n = |S|$: $p = |A|$

- Les seules opérations effectuées sont les opérations de coloriage, d'enfilement/défilement.
- Tout sommet passe dans la file :

Complexité

Grapge $G = (S, A)$, $n = |S|$: $p = |A|$

- Les seules opérations effectuées sont les opérations de coloriage, d'enfilement/défilement.
- Tout sommet passe dans la file :
  - soit parce qu'il est exploré dans `largeur_totale`

Complexité

Grapge $G = (S, A)$, $n = |S|$: $p = |A|$

- Les seules opérations effectuées sont les opérations de coloriage, d'enfilement/défilement.
- Tout sommet passe dans la file :
  - soit parce qu'il est exploré dans `largeur_totale`
  - soit parce qu'il est le voisin du sommet en tête de file.

Complexité

Grapge $G = (S, A)$, $n = |S|$: $p = |A|$

- Les seules opérations effectuées sont les opérations de coloriage, d'enfilement/défilement.
- Tout sommet passe dans la file :
  - soit parce qu'il est exploré dans `largeur_totale`
  - soit parce qu'il est le voisin du sommet en tête de file.
- Un sommet ne rentre jamais deux fois dans la file car il devient rouge dès sa première sortie de file. Or, ne rentrent dans la file que des sommets bleus.

Complexité

Grapge $G = (S, A)$, $n = |S|$: $p = |A|$

- Les seules opérations effectuées sont les opérations de coloriage, d'enfilement/défilement.
- Tout sommet passe dans la file :
  - soit parce qu'il est exploré dans `largeur_totale`
  - soit parce qu'il est le voisin du sommet en tête de file.
- Un sommet ne rentre jamais deux fois dans la file car il devient rouge dès sa première sortie de file. Or, ne rentrent dans la file que des sommets bleus.
- La coloration bleue initiale est en $O(n)$.
  Chaque sommet est exploré et tous ses voisins examinés (pour déterminer leurs couleurs).
  Comme la somme des nombres de voisins pour tous les sommets est p, la complexité est $O(n + p)$.

Exemple

Voir sur mes pages

Présentation

- Principe : on explore le graphe à partir d'un sommet x en visitant l'un de ses sommets successeurs y et en poursuivant l'exploration d'abord par les successeurs de ce dernier avant les autres successeurs de x.

Présentation

- Principe : on explore le graphe à partir d'un sommet x en visitant l'un de ses sommets successeurs y et en poursuivant l'exploration d'abord par les successeurs de ce dernier avant les autres successeurs de x.
- Ainsi l'exploration s'effectue en suivant le plus loin possible une chaîne issue de x. Lorsque tous les successeurs d'un sommet ont été visités, on continu l'exploration en remontant dans la chaîne au premier sommet ayant encore des successeurs non visités.

Présentation

- Principe : on explore le graphe à partir d'un sommet x en visitant l'un de ses sommets successeurs y et en poursuivant l'exploration d'abord par les successeurs de ce dernier avant les autres successeurs de x.
- Ainsi l'exploration s'effectue en suivant le plus loin possible une chaîne issue de x. Lorsque tous les successeurs d'un sommet ont été visités, on continu l'exploration en remontant dans la chaîne au premier sommet ayant encore des successeurs non visités.
- On gère une pile des sommets verts (bibliothèque OCAML listes ou `stack`, `python` : listes ou classe `deque`)

Algorithme

On utilise une pile pour gérer les sommets verts.

```
1  procedure Profondeur(G : graphe, P : pile)
2    Si P non vide
3       x := Peek P /*Récupère le sommet de la pile sans dépiler*/
4       tant qu'il existe un voisin bleu y de x :
5          Empiler ce voisin; :/* il devient vert*/
6          Profondeur(G,P)# on explore les descendants de y
7       Depiler x
8       Colorier x en rouge
9
10 Colorier tous les sommets en bleu
11 Créer une pile P vide /*Pile des sommets verts*/
12 tant que des sommets bleus sont présents faire
13     s := choisir un sommet bleu;
14     empiler s dans P; /* revient à colorier s en vert*/;
15     Profondeur(G,P)
```

En pratique

- Un même nœud s apparaît plusieurs fois au sommet de la pile. Il faut donc gérer un marqueur de progression dans sa liste de voisins pour éviter de reprendre cette liste depuis le début à chaque passage de s au sommet de la pile.

En pratique

- Un même nœud s apparaît plusieurs fois au sommet de la pile. Il faut donc gérer un marqueur de progression dans sa liste de voisins pour éviter de reprendre cette liste depuis le début à chaque passage de s au sommet de la pile.
- Solution : considérer les listes de voisins comme des piles. On dépile jusqu'à trouver un sommet bleu. Les voisins dépilés ne reviennent jamais dans la pile de voisins.
  Cela impose de faire une copie du graphe ($O(n)$ si le graphe est un tableau de listes de voisins.)

Exemple

Voir sur mes pages

Complexité

Complexité pour $G = (S, A)$: la même que celle du parcours en largeur.

- Chaque sommet est mis exactement une fois dans la pile.

Complexité

Complexité pour $G = (S, A)$: la même que celle du parcours en largeur.

- Chaque sommet est mis exactement une fois dans la pile.
- Pour chaque sommet, on examine tous ses voisins (c'est à dire tous les arcs sortant de ce sommet) une fois.

Complexité

Complexité pour $G = (S, A)$: la même que celle du parcours en largeur.

- Chaque sommet est mis exactement une fois dans la pile.
- Pour chaque sommet, on examine tous ses voisins (c'est à dire tous les arcs sortant de ce sommet) une fois.
- Au total $O(absS + |A|)$