INFO601 : graphes et algorithmes
TD 2 : parcours en profondeur, parcours en largeur

Pierre Hyvernat, Gérald Cavallini
Laboratoire de mathématiques de l'université de Savoie
bâtiment Chablais, bureau 17, poste : 94 22
email : Pierre.Hyvernat@univ-smb.fr
www : http://www.lama.univ-smb.fr/~hyvernat/

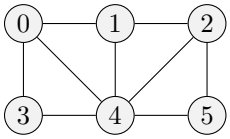
Exercice 1 : Parcours basique en profondeur

Rappel : on suppose que `G` est un graphe sous forme de listes d'adjacences. Le parcours en profondeur "basique" à partir du sommet `s0` est l'algorithme suivant, donné en Python

```
Vu = [False] * len(G)           # len(G) donne le nombre de sommets
Pred = [None] * len(G)

ATraiter = [s0]
while ATraiter:                 # tant que ATraiter est non vide
    s = ATraiter.pop()
    if Vu[s]: continue          # on ignore les sommets déjà vus
    Vu[s] = True
    for v in G[s]:              # pour tous les voisins ‘v‘ de ‘s‘
        if not Vu[v]:
            ATraiter.append(v)
            Pred[v] = s
```

Question 1. Faite tourner l'algorithme "à la main" sur le graphe suivant en partant de `v0` $= 0$, `v0` $= 1$ et `v0` $= 4$, Représentez graphiquement le tableau `Pred` obtenu.



Question 2. La liste `ATraiter` peut contenir des doublons. Pour conserver un parcours en profondeur tout en supprimant les doublons, il faudrait, avant d'ajouter un voisin avec `ATraiter.append(v)`, supprimer l'occurence éventuelle de `v` apparaissant dans `ATraiter`.

Qu'en pensez vous ?

Question 3. Quelle est la complexité au pire cas en temps de cet algorithme ?

Question 4. Cette version du parcours n'utilise aucune structure de donnée en dehors des tableaux (dynamiques) et peut donc se traduire dans n'importe quel autre langage.

En Python, on peut facilement remplacer
- le tableau `Pred` par un dictionnaire `Pred`
- le tableau `Vu` par un ensemble `Vu`

Quels sont les avantages et inconvénients de ces modifications ?

Question 5. Si le graphe n'est pas connexe, le parcours en profondeur basique ne parcourt pas tous les sommets. Modifiez légèrement le code pour qu'il relance la parcours autant de fois que nécessaire pour parcourir tous les sommets.

Remarque : dans ce cas, le tableau `Pred` ne contiendra plus un arbre, mais une *forêt*.

Exercice 2 : Parcours en largeur

Question 1. Le parcours en largeur basique s'obtient en changeant la pile ("last in, first out") `ATraiter` par une file ("first in, first out"), et en ne remettant pas les sommets avec un predecesseur dans `ATraiter`. Dans la boucle "`for v in G[s]`", le test devient donc

```
for v in G[s]:
if not Vu[v] and Pred[v] is None:
  ...
```

Faites tourner l'algorithme du parcours en largeur basique "à la main" sur l'exemple de l'exercice 1, en partant des sommets `v0` $= 0$, `v0` $= 1$ et `v0` $= 4$, Représentez graphiquement le tableau `Pred` obtenu.

Question 2. En Python, on peut simplement remplacer la ligne "`s = ATraiter.pop()`" qui enlève le dernier élément de `ATraiter` par "`s = ATraiter.pop(0)`" qui enlève le premier élément de `ATraiter` et on obtient ainsi un comportement de pile.

Que pensez vous de la complexité au pire cas de l'algorithme correspondant ?

Comment peut on améliorer les choses ?

Exercice 3 : parcours en profondeur récursif

Question 1. Il est parfois avantageux de faire un parcours en profondeur de manière récursive : pour faire un parcours à partir de "`s0`", on fait un parcours (récursivement) à partir de tous ses voisins.

Implémentez le parcours en profondeur récursif en (pseudo) Python. N'oubliez pas de gérer les sommets déjà vus pour éviter de boucler.

Question 2. Faites tourner l'algorithme sur les graphes de l'exercice 1, en partant de `v0` $= 0$, `v0` $= 1$ et `v0` $= 4$, Représentez graphiquement le tableau `Pred` obtenu.

Quel est le lien entre les résultats du parcours en profondeur itératif et ceux du parcours en profondeur récursif ?

Question 3. Un avantage du parcours récursif est que l'on sait quand on "commence" à traiter un sommet (début de la fonction récursive) et quand on "finit" de le traiter (fin de la fonction récursive). Au lieu du tableau de booléens `Vu`, on peut utiliser un tableau `Etat` qui enregistre une "couleur" pour chaque sommets :

- `VERT` (ou `PAS_VU`) pour les sommets que l'algorithme n'a encore jamais rencontré,
- `ORANGE` (ou `EN_COURS`) pour les sommets que l'algorithme a rencontré, mais n'a pas fini de traiter,
- `ROUGE` (ou `FINI`) pour les sommets que l'algorithme a complètement traité.

Quelles modifications doit on apporter au code de la question précédente pour conserver la couleur des sommets ?

Un graphe orienté contient un cycle (orienté) si et seulement si le parcours en profondeur récursif tombe sur un sommet `ORANGE`. Faites tourner cet algorithme à la main sur les graphes suivants, et implémentez l'algorithme en Python.