Devoir de théorie des graphes

Preuves d'algorithmes, et implémentations en Python

Dans ce devoir, il vous est proposé un code initial fonctionnel en Python, pour certains algorithmes du cours. Il vous sera demandé d'abord de compléter ce code, puis d'écrire de concevoir et implémenter d'autres routines.
Un travail de devoir à la maison ne vous sert à quelque-chose que si vous faites l'effort de le faire entièrement. Il n'est pas interdit de collaborer ou vous consulter; mais plus vous ferez de travail en autonomie, plus il vous sera profitable. Il vous est fortement recommandé de rédiger seul votre devoir. C'est l'occasion pour vous de garantir et d'approfondir votre compréhension du travail fourni pour le devoir.
Le document à rendre aura la forme d'un compte-rendu rédigé à l'ordinateur, et sera impérativement au format pdf. Les comptes-rendus sont à déposer sur la plateforme Chamilo dans l'espace MA351, au plus tard le ..... à minuit.
Prêtez attention au soin tant dans vos réalisations que dans vos explications. Un travail incomplet mais soigné pourra être mieux évalué qu'un travail apparemment complet mais mal réalisé et mal présenté.

1 Prise en main de Python

1.1 Le langage Python

Python est un langage de programmation et de scripting. Ca veut dire qu'il peut être utilisé à la place de langages comme C, C++ ou Java pour des développements assez conséquents; mais c'est aussi un remplaçant de Perl, dans son rôle de "couteau suisse du programmeur". Sa souplesse explique qu'il est beaucoup utilisé pour le Web côté serveur; mais ca n'est de loin pas son seul domaine d'application. Par exemple, on trouve beaucoup de code mathématique réalisé en Python. C'est vraiment un bon nouveau couteau suisse!
Les particularités de Python que vous devez connaître pour vous lancer sont les suivantes:

- C'est un langage dynamiquement typé. Cela signifie que les types des données sont attachés aux valeurs manipulées, et non pas aux variables, comme c'est le cas pour les langages statiquement typés. Vous pouvez donc affecter un nombre à une variable qui précédemment contenait une chaîne de caractères.

- Python est muni d'une bonne bibliothèque standard; elle est complétée par de nombreux paquets logiciels qui peuvent être téléchargés sur Internet.

- C'est un langage orienté objet. Certains types de données de la bibliothèque standard sont définis via un mécanisme de classes, qui supporte une notion d'héritage.

- Python a un système de modules assez propre; en cela, il ressemble plus à Java qu'à C.

- Python est un langage interprété. Cela signifie que vous pouvez charger votre code source dans un interprète pour le tester. Il est bien sûr possible de lancer une exécution en ligne de commande. Aucune compilation ou manipulation de fichiers objets, et aucune édition de lien ne sont nécessaires pour cela.

- Parce qu'il est possible de travailler dans l'interprète, l'utilisation de print() est rarement indispensable. L'expérience montre qu'utiliser des fonctions d'affichage rend le code moins souple et moins extensible. Il vous est recommandé d'essayer de coder "sans print()".

1.2 Préparation

Pour prendre en main Python, vous aurez besoin:

- d'un interprète Python. Ca se trouve dans toutes les bonnes crèmeries! Le code fourni fonctionne avec les version V2 et V3 de Python. Soyez informés que les syntaxes de ces deux langages sont (légèrement) incompatibles. La version 3 est la plus récente et la plus mûre; mais beaucoup de bibliothèques de code écrites par des tiers n'existent que pour Python V2.
- de documentations. Vous trouverez des documents de référence pour le langage, la bibliothèque standard ainsi que des tutoriels sur le site officiel du langage Python.

1.3 Démarrage de l'interprète

Votre première difficulté est de démarrer l'interprète dans un contexte où le code fourni vous est accessible. Sous linux, il suffit de lancer Python dans le répertoire où se trouve ce code.
Si c'est bien le cas, la commande `from graphes import *` devrait être acceptée silencieusement par Python.
`2+2` vous répond une valeur, alors que `a = 2+2` ne répond rien. Mais juste `'a'` vous donne la valeur attendue.
En tapant `(a, b) = 2+2, 3*3`, vous voyez comment Python vous permet de manipuler des tuples directement. Egalement avec `x = (2, 3)`: essayez ensuite `x[0]` pour récupérer les valeurs.
Les deux structures de données les plus utilisées dans Python sont les listes et les dictionnaires.

```
l = [1,2,3]
len(l)
l[0]
help(l)
type(l)
help(list)
dir(l)
d = {1:  'abc', 3:  'de'}
d[3]
d[2]
help(d)
dir(d)
```

2 Prise en main du code et réalisations

2.1 Fichiers fournis

L'archive qui est mise à votre disposition contient 3 fichiers sources Python: un générateur de graphes (graphes.py); la routine d'accessibilité, que nous verrons prochainement en cours, et qui est utilisée par le générateur d'arbres; et une implémentation de la promenade dans les arbres. Les générateurs prennent deux paramètres numériques. Le premier désigne le nombre de sommets du graphe à générer, mais il peut arriver que le graphe généré ait moins de sommets que ce qui a été demandé. Le second vous permet de générer différents graphes. La génération est déterministe: elle ne fait pas intervenir le hasard.

2.2 Premières réalisations dans les arbres

Après avoir tapé la commande d'import ci-dessus (`from graphes import *`), vous devez pouvoir générer un graphe G et le stocker dans une variable par l'appel suivant:
`g = graphe_oriente(100, 10)`

1. Quel est le format de stockage de ce graphe? Combien a-t-il de sommets? Quels sont les successeurs du sommet 1? Quels sont les successeurs du sommet 67?

2. Chargez maintenant le module promenade (`from promenade import *`).

Examinez le graphe produit par la commande `arbre(10, 4)`.
C'est un graphe non-orienté, ce qui veut dire que vu comme un graphe orienté, à chaque fois que (x, y) est un arc du graphe, (y, x) en est également un.
De plus, c'est un arbre: ceci signifie que le graphe est connexe et sans cycle.
Donnez-en une représentation graphique, en tant que graphe non-orienté, puis en tant que graphe orienté.

3. Ecrivez une fonction Python "`deux_feuilles()`", qui prend en entrée un arbre, que l'on supposera avoir au moins deux sommets, et renvoie deux feuilles distinctes de cet arbre.

4. Ecrivez une routine `decompose(g)` de décomposition d'un arbre. Elle prend en entrée un arbre, et retourne d'une part un sommet x, et d'autre part une liste de couples $[(u_1, v_1), (u_2, v_2), \ldots, (u_k, v_k)]$ telle que l'arbre donné en entrée puisse être reconstruit à partir du sommet x en joignant une nouvelle feuille v_1 au sommet u_1, puis une nouvelle feuille v_2 au sommet u_2, et ainsi de suite.

5. Ecrivez une routine de mise en forme du résultat de la routine précédente. Voici un exemple de sortie possible pour l'arbre obtenu par `arbre(4, 1)`, c'est-à-dire:
`{1: [2, 3, 4], 2: [1], 3: [1], 4: [1]}`:

En partant du sommet 2,
attacher la feuille 1 au sommet 2,
attacher la feuille 4 au sommet 1,
attacher la feuille 3 au sommet 1,

6. Ecrivez une routine qui détermine une coloration en 2 couleurs de l'arbre passé en argument. Testez-là sur l'arbre de la question 2.
Indication: vous pourrez vous appuyer sur le travail fourni à la question 4.

2.3 Circuits et tris topologiques

7. Créez un nouveau module pour la recherche de circuits et de tris topologiques. Y placer une routine qui trouve dans un graphe orienté non vide un puits ou un circuit. Assurez-vous que cette routine retourne bien soit un puits, soit un circuit.

8. Implémentez une fonction qui prend un graphe orienté en paramètre, et retourne soit un circuit du graphe, soit un tri topologique du graphe.

2.4 Graphes réguliers de degré 2

On appelle graphe régulier un graphe dont tous les sommets ont le même degré. Un graphe régulier de degré k est un graphe dont tous les sommets sont de degré k.

9. Créez un nouveau module pour cette partie. Ecrire une routine qui détecte un cycle dans un graphe régulier de degré 2.

10. Ecrire une routine qui révèle la structure en cycles d'un graphe régulier de degré 2.
Testez cette routine sur au moins 2 graphes.