Projet de théorie des graphes en Python :

Perdu dans le labyrinthe !

Yann Kieffer, Vincent Guisse - Grenoble INP-Esisar, UGA

La Prépa des INP, Valence

Sommaire

1 Contexte, objectifs

Il vous est proposé dans ce projet de mobiliser les notions vues en cours et en TDs d'arbres et de graphes, afin de résoudre en pratique des problèmes de cheminement dans des labyrinthes.

Le projet consiste en la réalisation de plusieurs programmes en Python, dont certains pourront mobiliser des routines déjà réalisées lors des séances de TD.

La thématique générale est celle du labyrinthe. Dans un labyrinthe, nous voyons ce qui est immédiatement autour de nous ; bien souvent, nous n'avons pas le plan du labyrinthe. Pour cheminer dans le labyrinthe, nous pouvons avancer pas à pas, découvrant ce faisant d'autres lieux du labyrinthe.

Cette situation peut être modélisée par un graphe. Dans le cadre de ce projet, nous utiliserons des graphes orientés : ils permettent aussi de modéliser des situations où les graphes non-orientés auraient pu être utilisés.

Ce type de modélisation est utilisé par certains types de jeux vidéos, dits "jeux d'aventure" : vous découvrez le monde du jeu au fur et à mesure de votre évolution dans celui-ci.

Les objectifs pédagogiques de ce projet sont les suivants :

- Réaliser des programmes complets en Python, dont la fonctionnalité peut être comprise par des non-spécialistes de l'informatique.
- Mobiliser des algorithmes pour concrétiser des possiblités de résolution automatique.
- Faire communiquer entre eux des programmes Python.

2 Organisation

Pour la réalisation de ces projets, nous vous invitons à travailler en binômes. La constitution des binômes est libre.

Nous avons 3 séances encadrées dédiées à la réalisation des projets :

- Mercredi 27 novembre, 10h
- Mercredi 4 décembre, 8h15
- Mercredi 11 décembre, 10h

Il est possible, et encouragé, de travailler sur le projet entre les séances. Même un temps court passé sur le projet entre deux séances vous fera gagner du temps en séance.

Parfois, plusieurs temps courts valent mieux qu'un temps long !

3 Evaluation

Les projets seront évalués le 18 décembre 2025, entre 10h et 11h30. Lors de l'évaluation, vous présenterez vos projets à l'oral aux deux encadrants des projets, sans autre support que le code informatique produit. Vous pourrez faire des démonstrations. Vous devrez répondre aux questions des encadrants, qui pourront aussi vous demander à voir certains de vos programmes Python.

Le sujet indique clairement quelles parties sont obligatoires, et quelles parties sont optionnelles. Vous pouvez déborder du sujet autant que vous voudrez.

Les soutenances ne dureront pas plus de 10 minutes.

La note attribuée à la soutenance sera la note finale du cours pour la partie arbres/graphes.

4 Méthodes de travail

Voici quelques recommandations générales pour développer du code informatique à plusieurs.

1. N'hésitez pas à écrire des fonctions utilitaires, qui pourraient vous servir dans l'écriture d'autres fonctions.
2. Rangez vos fonctions dans des modules, en les regroupant de manière logique. Evitez de faire des modules trop touffus.
3. Pensez bien à documenter chaque fonction définie, en précisant le nombre et la nature arguments passés en paramètre, et la présence et la nature éventuelle de la valeur de retour.
4. Pour chaque fonction codée, testez-là avant de l'intégrer avec les autres, et archivez le code de test de manière à pouvoir repasser les tests efficacement. Repasser un test est particulièrement utile et commode en cas de modification du code.
5. Définissez un protocole pour la collaboration, en particulier pour les séances de travail individuelles. Parmi les possibles :
   — se limiter à modifier uniquement des fichiers bien définis ;
   — se maintenir informé en début et fin de séance de travail des fichiers qui vont être modifiés, et transmettre les nouvelles versions.

5 Fichier fourni

Il vous est fourni un unique fichier, `exemple.py`, contenant un unique graphe d'exemple. Il illustre le format d'encodage à utiliser, et peut vous aider à déboguer vos routines.

6 Travail à effectuer

6.1 Se promener dans un labyrinthe

Le premier programme à réaliser est un programme de cheminement dans un graphe.

Les graphes que vous manipulerez dans le cadre de ce projet ont un format très précis. Ils contiennent toutes les informations attendues dans un graphe présenté sous forme de listes de successeurs, et quelques informations en plus. Vous pouvez consulter le graphe d'exemple fourni.

Les informations supplémentaires sont :

— Une étiquette pour chaque sommet du graphe.
— Une étiquette pour chaque arc du graphe.

Par étiquette, nous entendons une chaîne de caractères décrivant l'objet auquel elle est attachée.

Le programme à réaliser prendra deux paramètres : un graphe, et un sommet du graphe, qui sera le sommet de départ. Il aura le comportement suivant : tant que le sommet courant a des successeurs, le programme exécute la séquence d'instructions suivante :

— Afficher l'étiquette du sommet courant ;
— inviter l'utilisateur à indiquer par où il veut aller ensuite ;
— lister, en les numérotant, toutes les options possibles : pour chaque option (=arc), la seule information communiquée à l'utilisateur sera l'étiquette associée à cet arc ;
— saisir une chaîne de caractères sur l'entrée standard ;

— vérifier que la chaîne encode bien un nombre ; sinon, inviter l'utilisateur à faire une nouvelle saisie ;
— vérifier que le nombre correpond à une option proposée ; sinon, inviter l'utilisateur à faire une nouvelle saisie ;
— mettre à jour le sommet courant.

Lorsque le sommet courant n'a plus de successeurs, on considère que l'utilisateur est sorti du labyrinthe : on peut le féliciter pour cela.

Réaliser une fonction Python qui implémente cette routine : vous pourrez la nommer `visite()`.

6.2 Peut-on sortir du labyrinthe ?

Vous avez peut-être accès au plan du labyrinthe : peut-être êtes-vous en communication avec son créateur ? Ou peut-être que son créateur, c'est vous ?

Créez maintenant un nouveau programme Python, qui prenne en entrée un graphe, un sommet de départ, et un sommet d'arrivée, et qui calcule la succession des sommets pour aller du départ à l'arrivée. Ce programme doit émettre un message d'erreur si la sortie n'est pas accessible depuis l'entrée.

Enrichissez ce programme d'une seconde routine, qui prenne en entrée un graphe et un chemin du graphe, et qui calcule la suite des ordres à donner à la routine `visite()` afin de sortir du labyrinthe. Ces ordres seront indiqués sur la sortie standard, un par ligne.

Branchez la sortie de ce programme sur celui effectuant la visite, en utilisant la construction dite "pipe" dans un interprète de commande :

```
prog1 | prog2
```

Vérifiez que cela fonctionne bien !

6.3 Un plan pour sortir

En mobilisant les outils de tracé de graphe présentés dans la partie du cours portant sur les arbres, produisez un plan permettant à tout un chacun de rejoindre n'importe quel sommet acessible du labyrinthe, à l'aide du programme `visite()` et de ce plan.

6.4 A-t-on raté la sortie ?

Dans certains contextes de parcours de labyrinthe, il est possible de se retrouver à un endroit à partir duquel il n'est plus possible de sortir. Cela peut être le cas par exemple dans les "livres dont vous êtes le héros". Avez-vous déjà essayé d'y jouer ?

Dans cette partie, vous allez réaliser un programme qui détermine l'ensemble des sommets à partir desquels il est encore possible de sortir du labyrinthe.

Commencez par écrire une routine qui détermine toutes les sorties possibles du labyrinthe. Rappelez-vous qu'une sortie est un sommet sans successeur.

Plutôt que de chercher si une sortie est accessible pour chaque sommet du graphe, nous vous proposons la démarche suivante.

Créer une "sortie unique", en rajoutant un nouveau sommet, qui sera successeur de toutes les sorties identifiées. (Pensez à copier le graphe, sans quoi vos modifications modifieraient aussi le graphe qui vous est passé en paramètre.)

Utiliser la routine `accessibles()` sur le graphe miroir, à partir d'un sommet bien choisi, pour déterminer l'ensemble des sommets du graphe à partir desquels il est possible de sortir du labyrinthe.

Ecrire maintenant une routine qui calcule l'ensemble des sommets à partir desquels il n'est pas possible de sortir du labyrinthe.

7 Extensions facultatives

7.1 Un éditeur de labyrinthe

Imaginez, puis réaliser un éditeur de labyrinthe, permettant à une personne pas familière de Python d'encoder de nouveaux labyrinthes pour les utiliser dans vos programmes.

7.2 Sortir au plus vite !

Imaginez un formalisme enrichi pour les labyrinthes, permettant d'encoder pour chaque arc une longueur.

Utilisez un algorithme de plus court chemin pour déterminer comment s'échapper au plus vite du labyrinthe.
