Travaux pratiques - Graphes et algorithmes

Exercice 1 Algorithmes - Parcours de graphes

Objectifs

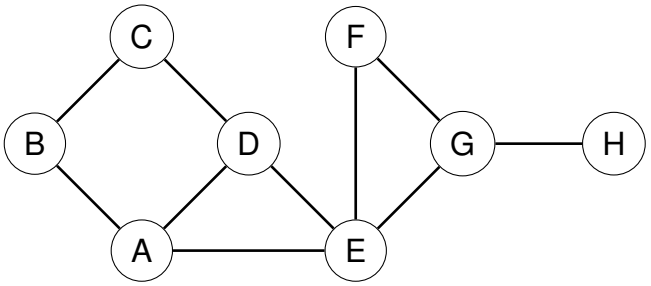
- Etre capable de représenter un graphe à partir d'une liste d'adjacences
- Implémenter en Python des algorithmes de parcours de graphes et de recherche de chemins dans un graphe.
- Utiliser ces fonctions Python pour résoudre un problème utilisant des graphes (labyrinthes,mots voisins ...)

Chaque fonction doit être documentée en utilisant les docstring et incluant plusieurs jeux de tests utilisant la bibliothèque `doctest`.

Les structures de pile et file utilisées dans ce TP seront imlplémentées à l'aide du type `List` de python et des méthodes `pop` et `append` associées.

PARTIE A - Structures de données pour représenter un graphe

1. Proposez une structure de données Python pour représenter le graphe suivant par une liste d'adjacence.



PARTIE B - Parcours d'un graphe

Parcours en largeur d'abord

Voici la traduction en pseudo-code du parcours en largeur d'abord d'un graphe.

```
F est une file vide
On enfile un sommet
Tant que F n'est pas vide
        S = Tete de F
        On enfile les voisins de S qui ne sont pas deja presents dans la file
            et qui n'ont pas ete deja defiles
        On defile F
Fin Tant Que
```

2. En partant du sommet A, écrire la liste des sommets d'un parcours en largeur d'abord du graphe de la partie A.
3. Ecrire le code d'une fonction Python nommée `parcours_en_largeur(graphe,sommet)` qui parcourt en largeur un graphe à partir du sommet `sommet` et dont sa liste d'adjacence `graphe` est représentée par un dictionnaire.
   Cette fonction retournera la liste des sommets parcourus si le sommet `sommmet` appartient bien au graphe et `None` sinon.

4. Testez votre fonction avec le graphe de la partie A et avec ”A” comme sommet de départ.
5. Comparer la liste retournée par la fonction avec le résultat de la question 2 ?
6. Quelle remarque peut-on formuler sur l'ordre des sommets affichés ?

Parcours en profondeur

Voici la traduction en pseudo-code du parcours en profondeur d'un graphe.

```
P est une pile vide
On empile un sommet
Tant que P n'est pas vide
        S = depile(P)
        On empile les voisins de P qui ne sont pas deja presents dans la pile
           et qui n'ont pas ete deje depiles
Fin Tant Que
```

7. En partant du sommet A, écrire la liste des sommets d'un parcours en profondeur du graphe de la partie A.
8. Ecrire le code d'une fonction Python nommée `parcours_en_profondeur(graphe,sommet)` qui parcourt en profondeur un graphe à partir du sommet `sommet` et dont sa liste d'adjacence `graphe` est représentée par un dictionnaire.
   Cette fonction retournera la liste des sommets parcourus si le sommet `sommmet` appartient bien au graphe et `None` sinon.
9. Testez votre fonction avec le graphe de la partie A et avec ”A” comme sommet de départ.
10. Comparer la liste retournée par la fonction avec le résultat de la question 2 ?
11. Quelle remarque peut-on formuler sur l'ordre des sommets affichés ?

PARTIE C - Recherche de chemins dans un graphe

Le loup,la chèvre,un chou et un passeur ....
Sur la rive d'un fleuve se trouvent un loup,une chèvre,un chou et un passeur. Le problème consiste à tous les faire passer sur l'autre rive à l'aide d'une barque,menée par le passeur,en respectant les règles suivantes :

- la chèvre et le chou ne peuvent pas rester sur la même rive sans le passeur ;
- la chèvre et le loup ne peuvent pas rester sur la même rive sans le passeur ;
- le passeur ne peut mettre qu'un seul ”passager” avec lui.

On décide de représenter le passeur par la lettre P,la chèvre par la lettre C,le loup par L et le chou par X.

12. Représenter ce problème à l'aide d'un graphe où les sommets sont tous les états possibles sur la rive de départ (par exemple,”PLCX” est un sommet représentant le fait que tous sont sur la rive de départ.)
13. Trouver une solution au problème en indiquant chacun des déplacements (si possible une solution avec le moins de déplacements possibles).

Algorithmes de recherche d'un chemin entre deux sommets

14. Implémentez en Python une fonction Python nommée `cherche_chemin(graphe,depart,arrivee` qui recherche et retourne un chemin (s'il existe) entre les sommets `depart` et `arrivee` dans le graphe `graphe` à partir de l'algorithme suivant :

```
Fonction cherche_chemin(graphe,depart,arrivee)
        P <--pile vide
        empiler le couple (depart,[depart]) dans P
        Tant que P n'est pas vide faire
                (sommet,chemin) <-- tete de P
                listes_nouveaux_sommets_voisins <-- liste des sommets adjacents
                    a sommet qui ne sont pas dans chemin
                Pour un_sommet dans listes_nouveaux_sommets_voisins
                        Si un_sommet = arrivee alors
                                retourner chemin + [un_sommet]
                        sinon
                                empiler (un_sommet,chemin + [un_sommet])
                        FinSi
                FinPour
        FinTantQue
FinFonction
```

15. Testez la fonction Python `cherche_chemin(graphe,depart,arrivee)` avec le graphe de la partie A.
16. Vérifiez si les chemins proposés sont de plus courte distance. Si ce n'est pas le cas,citez des chemins donnés par la fonction Python qui ne sont pas de plus courte distance.
17. A partir de la fonction précédente, proposez une autre fonction `cherche_tous_chemins(graphe,de` qui retourne la liste de tous les chemins entre deux sommets.
18. Complétez la fonction qui retourne la liste de tous les chemins entre deux sommets pour créer une fonction nommée `cherche_chemin_plus_court(graphe,depart,arrivee)` qui retourne le plus court chemin entre deux sommets.

PARTIE D : Cycle d'un graphe

19. Donnez trois exemples de cycle dans la graphe de la partie A ayant pour sommets de départ les sommets A, E et G.
20. Proposez le code d'une fonction nommée `cherche_cycles(graphe,sommet)` qui retourne la liste de tous les cycles de sommet `sommet` du graphe `graphe`. Il serait judicieux que cette fonction appelle la fonction `cherche_tous_chemins(graphe, depart,arrivee)`.
21. Testez votre fonction avec le graphe de la partie A et comparer avec les résultats de la question précédente.
22. Proposez le code d'une fonction nommée **cherche_tous_cycles(graphe)** qui retourne la liste de tous les cycles du graphe `graphe`. Il serait judicieux que cette fonction appelle la fonction `cherche_tous_chemins(graphe, depart,arrivee)`.

Exercice 2 Algorithme PageRank

Objectifs

- Modéliser un réseau de pages Web liées entre elles par des liens hypertextes à l'aide de graphe.
- Simuler un parcours aléatoire dans ce graphe ainsi modélisé pour classer les pages Web selon un critère de "popularité" imitant ainsi l'algorithme du moteur de recherche PageRank de Google

Présentation de l'algorithme PageRank

PageRank est l'algorithme d'analyse des liens concourant au système de classement des pages Web utilisé par le moteur de recherche Google. Il mesure quantitativement la popularité d'une page web. Le PageRank n'est qu'un indicateur parmi d'autres dans l'algorithme qui permet de classer les pages du Web dans les résultats de recherche de Google.

Ce système a été inventé par Larry Page, cofondateur de Google. Ce mot est une marque déposée. Le théorème de point fixe est le concept mathématique qui a rendu possible le calcul du PageRank.

Le principe de base est d'attribuer à chaque page une valeur (ou score) proportionnelle au nombre de fois que passerait par cette page un utilisateur parcourant le graphe du Web en cliquant aléatoirement, sur un des liens apparaissant sur chaque page.
Ainsi, une page a un *PageRank* d'autant plus important qu'est grande la somme des PageRanks des pages qui pointent vers elle (elle comprise, s'il y a des liens internes).

Le PageRank est une mesure de centralité sur le réseau du web. Plus formellement, le déplacement de l'utilisateur est une **marche aléatoire sur le graphe du Web**, c'est-à-dire le **graphe orienté dont les sommets représentent les pages du Web et les arcs les hyperliens**.
En supposant que l'utilisateur choisisse chaque lien indépendamment des pages précédemment visitées, il s'agit d'un processus de Markov.
Le PageRank est alors simplement la probabilité stationnaire d'une chaîne de Markov.

Simulation de PageRank en Python

Vous allez simuler le parcours aléatoire d'un graphe simulant un mini-Web par un internaute et classer les pages Web avec un score de popularité (à la PageRank) à l'aide d'un programme développé en Python.

Le mini-Web est modélisé par un ensemble de pages Web qui sont reliées entre elles par des liens hypertextes.

Ce mini-Web est représenté par un **graphe orienté, non pondéré dont les sommets sont les noms des pages Web et les arêtes les liens hypertextes**.

On modélisera ce graphe par une **liste d'adjacence** avec comme structure de données en Python un dictionnaire dont :

- les clefs sont les sommets du graphe (les noms des pages Web)
- les valeurs la liste des pages Web liées à une page donnée par un lien hypertexte.

1. Complétez la fonction Python nommée `genere_mini_web_aleatoire(nb_pages)` qui génère aléatoirement un mini-Web composé de `nb_pages` pages Web reliées entre elles par des liens hypertexte.

```
def genere_mini_web_aleatoire(nb_pages):
    mini_web = {f'Page{i}': [] for i in range(nb_pages)}

    for i in range(nb_pages):
        num_links = random.randint(...)  # Each page will have at least one
            link
        ...
          links = ...
          ...
        mini_web[f'Page{i}'] = ...

    return mini_web
```

```
# Exemple d'utilisation
mini_web = genere_mini_web_aleatoire(5)
print(mini_web)
#OUTPUT
{'Page0': ['Page2', 'Page1', 'Page4', 'Page3'], 'Page1': ['Page3'], 'Page2': ['
   Page1', 'Page0', 'Page4', 'Page3'], 'Page3': ['Page0'], 'Page4': ['Page2',
   'Page1']}
```

2. Visualiser le graphe ainsi modélisé à l'aide de la bibliothèque Python Graphviz et de la fonction suivante : FinFonction

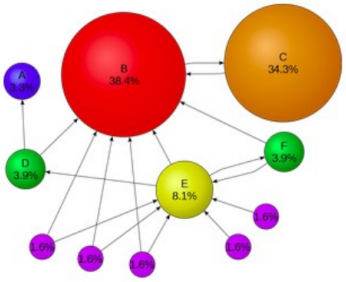
```
def visualiser_graphe(mini_web):
    dot = graphviz.Digraph(comment='Mini-Web')

    for page, links in mini_web.items():
        dot.node(page)
        for link in links:
            dot.edge(page, link)

    return dot
```

3. Ecrire une fonction Python nommée `simule_marche_aleatoire(miniWeb,nb_deplacements)` qui génère aléatoirement un mini-Web composé de `nb_pages` qui simule une marche aléatoire d'un internaute parmi le mini-Web représenté par le graphe miniWeb composée de `nb_deplacements` déplacements sur ce graphe.
   Cette fonction devra retourner la liste des pages Web du graphe classées par ordre de popularité en se basant sur le principe que plus une page est visitée et plus elle est populaire.

4. Représenter graphiquement le mini-Web avec ses pages classées par ordre de popularité à l'aide de la bibliothèque Python Graphviz. La taille (ou la couleur) d'un disque représentant une page sera proportionnelle à sa popularité.



Sources :

- Hugues Malherbe