Exploration de graphes

1 Introduction

On s'intéresse à un graphe non-orienté issu d'un jeu (édité par Ravensburger en France), appelé `Scotland Yard`, modélisant la ville de Londres. Il contient 199 sommets, numérotés de 1 à 199. Chaque arête du graphe est coloriée de trois couleurs différentes : jaune, turquoise et rouge (représentant respectivement des trajets effectués en taxi, en bus ou en métro). On notera qu'il est parfois possible de trouver plusieurs arêtes adjacentes à deux mêmes sommets i et j de couleurs différentes.

Le graphe est fourni sous la forme d'un tableau `liens` de tableaux de listes, de sorte que `tab.(i).(0)` donne les voisins du sommet i pour les arêtes jaunes, `tab.(i).(1)` pour les arêtes turquoises, `tab.(i).(2)` pour les arêtes rouges. Dans chaque liste, les voisins sont ordonnés par ordre croissant. On notera la présence d'une case d'index 0 dans le tableau, non utiliée, pour que la numérotation corresponde à celle du plateau. Dans tout le sujet, on considérera que `tab` est une variable globale pour ne pas avoir à la fournir à toutes les fonctions.

2 Opérations élémentaires

1. Proposer une fonction `union` de signature `'a list -> 'a list -> 'a list` prenant en argument deux listes triées par ordre croissant sans doublon, et renvoyant la liste correspondant à leur union, également triée par liste croissant sans doublon.

2. En déduire une fonction `voisins` de signature `int -> int list` prenant en argument un sommet $s \in [\![1 \,..\, 199]\!]$ et renvoyant la liste de ses voisins sans doublons, en ne tenant pas compte de la couleur.

3. Déterminer la liste des sommets de plus grand degré du graphe ainsi que leur degré.

4. Proposer une fonction `chemin` de signature `int -> int -> int list` prenant en argument deux sommets s et s' et renvoyant une liste de sommets représentant un chemin de s à s' (la liste devra commencer par s, se terminer par s', et toute paire de sommets consécutifs dans la liste doit correspondre à une arête). On demande un chemin quelconque, pas nécessairement un plus court chemin. On pourra se baser sur les idées d'un parcours en profondeur et de retour sur trace.

3 Propriétés du graphe

On définit l'excentricité d'un sommet s, dans un graphe non-orienté connexe, comme le maximum parmi les distances à l'ensemble des sommets s' du graphe.

5. En utilisant un parcours adéquat, proposer une fonction `excentricite` de signature `int -> int`, prenant en argument un sommet s et renvoyant l'excentricité de s. Déterminer l'excentricité du sommet 42.

On définit le rayon d'un graphe comme la plus petite excentricité d'un sommet du graphe, et les sommets dont l'excentricité est égale au rayon comme les centres du graphe.

6. Proposer une fonction `rayon` de signature `unit -> int` ne prenant aucun argument et renvoyant le rayon du graphe. Quel est le rayon du graphe étudié ?

7. Proposer une fonction `centres` de signature `unit -> int list` ne prenant aucun argument et renvoyant les centres un graphe. Combien y en a-t-il dans le graphe considéré ?

On définit le diamètre d'un graphe comme la plus grande excentricité d'un sommet du graphe.

8. Proposer une fonction `diametre` de signature `unit -> int` ne prenant aucun argument et renvoyant le diamètre du graphe. Quel est le diamètre pour le graphe étudié ? Est-il le double du rayon ?

9. Proposer une fonction `distants` de signature `int -> int -> int list` prenant en argument un sommet s et une distance d et renvoyant la liste de sommets à une distance d de s.

10. Proposer une fonction `plus_loins` de signature `unit -> (int * int) list` renvoyant la liste sans doublons de toutes les paires non-orientées de points dont la distance est égale au diamètre du graphe.

4 Chemins alternants

On cherche à présent à déterminer s'il existe un chemin de i à j tel que les arêtes suivies alternent les couleurs dans cet ordre : jaune, turquoise, rouge, jaune, ...

On souhaite construirre une fonction `chemin_alternant` de signature `int -> int -> int list` (fonctionnant comme la fonction `chemin` précédente mais avec la contrainte supplémentaire) répondant à ce problème. On pourra passer plusieurs fois par le même sommet, voire par la même arête.

11. Justifier que s'il existe un tel chemin, alors il existe un chemin tel que l'on ne visite pas deux fois le sommet i en venant d'une arête de même couleur.

12. En déduire que l'on peut chercher un chemin normalement dans un graphe G′ bien choisi.

13. Proposer une implémentation de la fonction `chemin_alternant`.

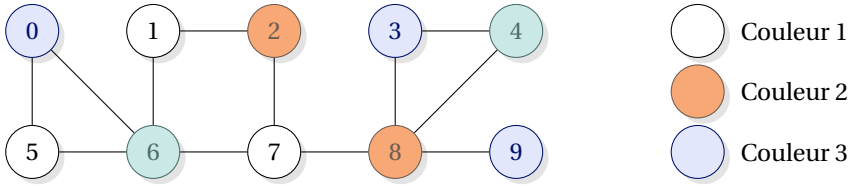
5 Chemins de longueur donnée

On cherche enfin à déterminer s'il existe un chemin de i à j de longueur $d > 0$ donnée ne passant pas deux fois par le même sommet (excepté possiblement le premier et dernier sommets, de sorte qu'il soit possible de rechercher des cycles). On cherche donc une séquence de sommets $i = s_0 \rhd s_1 \rhd \ldots \rhd s_{d-1} \rhd s_d = j$ tels que tous les s_i pour $i > 0$ soient distincts, et (s_{i-1}, s_i) soit une arête du graphe pour tout $i > 0$.

Il s'agit d'un problème bien plus difficile. On propose ici une solution dite « de type Las Vegas », qui fonctionne avec une composante aléatoire, et qui, si elle réussit, fournit un tel chemin, mais ne peut pas garantir une réussite à chaque tentative.

Pour ce faire, on attribue à chaque sommet s_i pour $i \in [\![1 \,..\, 199]\!]$ une couleur aléatoire codée par un entier entre 1 et $d-1$. Ces couleurs sont mémorisées dans un tableau `c` (de type `int array`) de longueur n (l'entier `c.(s)` désigne ainsi la couleur du sommet s). On impose par ailleurs `c.(j)=d` (la couleur de i est sans importance, ce qui permettra de rechercher des cycles).

Par exemple, dans le graphe suivant, il existe un chemin menant de $i = 6$ à $j = 4$ pour $d = 4$ et respectant la coloration, le chemin $6 \rhd 7 \rhd 8 \rhd 3 \rhd 4$:



Pour mémoriser une coloration de notre graphe à 199 sommets, on utilisera un tableau de taille 200, la case d'index 0, non utilisée, pouvant contenir une valeur quelconque.

On dispose d'une fonction `Random.int` prenant en argument un entier p et retournant un entier choisi aléatoirement entre 0 et $p-1$ (inclus) avec une loi uniforme (c'est-à-dire que la probabilité d'obtenir un entier $j \in [\![0 \,..\, p-1]\!]$ est $1/p$).

14. Proposer une fonction `coloriage_aleatoire` de signature `int -> int -> int array` prenant en argument un sommet j, une distance d, et renvoyant un coloriage aléatoire selon les paramètres décrits précédemment.

15. Écrire une fonction `voisins_de_couleur` de signature `int -> int array -> int list` prenant en argument un sommet s, une coloration et un entier k et renvoyant la liste des sommets voisins de s dans le graphe dont la couleur est k.

16. Écrire une fonction `voisins_de_la_liste_de_couleur` de signature `int list -> int array -> int list` prenant en argument une liste de sommets s_i, une coloration et un entier k et renvoyant la liste sans redondance des sommets voisins des s_i dans le graphe dont la couleur est k.

17. Quelle serait, dans un cadre plus général, la complexité dans le pire cas de la fonction précédente en fonction de l'ordre n du graphe et du nombre p d'arêtes dans le graphe ?

18. En déduire une fonction `chemin_arc_en_ciel` de signature `int -> int -> int -> int list option` qui prend en argument i, j et d, effectue un coloriage aléatoire et renvoie `None` si elle échoue à trouver un chemin de i à j avec l'approche précédente, ou `Some lst` où `lst` est une liste de longueur $d+1$ de sommets représentant un chemin de i à j si elle il parvient.

19. Quelle est, dans un cas général, la complexité dans le pire cas de la fonction précédente en fonction de n, p et d ?

Si les couleurs des sommets sont choisies aléatoirement et uniformément dans $[\![1 \,..\, d-1]\!]$, la probabilité que k soit la couleur du k^{e} sommet d'une suite fixée de $d-1$ sommets vaut $1/d-1$ indépendamment pour tout k. Ainsi, étant données deux sommets i et $j \in [\![1 \,..\, n]\!]$, s'il existe dans le graphe un chemin propre de i à j de longueur d, la fonction `chemin_arc_en_ciel` retourne un chemin avec une probabilité au moins $(d-1)^{-d+1}$; et répond toujours `None` sinon.

Si un tel chemin existe, la probabilité qu'une parmi $(d-1)^{-d+1}$ exécutions indépendantes de `chemin_arc_en_ciel` trouve un chemin est donc supérieure ou égale (admis) à

$$1 - \left(1 - (d-1)^{-d+1}\right)^{(d-1)^{-d+1}} = 1 - \exp\left((d-1)^{-d+1} \ln\left(1 - (d-1)^{-d+1}\right)\right) \geqslant 1 - 1/e > 0$$

20. Écrire une fonction `chemin_arc_en_ciel_multi` de même signature que `chemin_arc_en_ciel` et renvoyant,avec une probabilité d'au moins $1 - 1/e$, un chemin adéquat s'il en existe un de i à t, et renvoie toujours `None` sinon.