TP

Révision sur les graphes

Ce sujet à pour but de vous faire réviser deux notions essentielles sur les graphes vues en première année : représentation par matrices ou listes d'adjacence et parcours d'un graphe.

1. Génération pseudo-aléatoire d'une suite d'entiers

On définit une suite (u_k) par la donnée de la condition initiale $u_0 = 42$ et la relation de récurrence

$$\forall k \in \mathbb{N}, \quad u_{k+1} = 1\,022 \times u_k \bmod 65\,533$$

Cette suite sera utilisée pour générer des graphes pseudo-aléatoires. Pour que la création de ces derniers ne soit pas trop longue, il est indispensable de stocker dans une liste les différentes valeurs de u_k, pour $0 \leqslant k < 100\,000$.

Question 1. Définir et remplir cette liste, puis donner les valeurs suivantes :

a) $u_{10} \bmod (1\,000)$ b) $u_{500} \bmod (1\,000)$ c) $u_{10\,000} \bmod (1\,000)$

2. Représentation d'un graphe par matrice d'adjacence

Graphe orienté

Un graphe orienté est un couple $G = (V, E)$ où V est un ensemble fini de sommets et $E \subset V \times V$ un ensemble d'arcs. Lorsque $(i, j) \in E$, i est appelé l'*origine* et j la *cible* de l'*arc* (i, j).
On notera qu'avec cette convention il est possible d'avoir $(i, i) \in E$, autrement dit d'avoir un arc qui relie un sommet à lui-même. D'autres conventions peuvent exclure cette situation.

Sans perte de généralité on supposera pour la suite $V = [\![0, n-1]\!]$, n désignant le nombre de sommets de G.

Représentation par matrice d'adjacence Elle consiste à représenter le graphe $G = (V, E)$ par la matrice $M \in \mathcal{M}_n(\{0, 1\})$, avec

$$M_{i,j} = \begin{cases} 1 & \text{si } (i,j) \in E \\ 0 & \text{sinon} \end{cases}$$

Attention. Contrairement aux conventions mathématiques, les lignes et les colonnes de cette matrice seront indexées par des entiers compris entre 0 et $n-1$.

Par la suite on note $G_{n,k,p}$ le graphe de sommets $V = [\![0, n-1]\!]$, tel que $(i, j) \in E$ si et seulement si $u_{i+u_{k+j}}$ est divisible par p.

Question 2. Définir une fonction `graphe_M(n, k, p)` qui renvoie la matrice d'adjacence du graphe $G_{n,k,p}$. On utilisera la syntaxe ci-dessous pour définir une matrice à n lignes et n colonnes remplies de 0 :

```
M = [[0 for j in range(n)] for i in range(n)]
```

Question 3. Étant donné un sommet v, on appelle degré *entrant* de v le nombre d'arcs dont v est la cible.
Calculer le maximum des degrés entrants des sommets de $G_{n,0,p}$ pour les valeurs suivantes :

a) $(n, p) = (100, 2)$ b) $(n, p) = (100, 10)$ c) $(n, p) = (100, 50)$

3. Représentation d'un graphe par liste d'adjacence

La représentation d'un graphe par matrice d'adjacence a une complexité spatiale en $O(n^2)$ (le nombre de cases de la matrice M), où $n = \operatorname{card} V$. Lorsque le nombre d'arcs est faible devant n^2, ce mode de représentation n'est pas très économique, et on lui préfère alors une représentation par liste d'adjacence, dont la complexité est un $O(n + m)$ avec $m = \operatorname{card}(E)$: on représente le graphe G par une liste de listes L, où pour tout $i \in [\![0, n-1]\!]$, $L[i]$ est la liste des sommets accessibles à partir de i.

Question 4. Définir une fonction `graphe_L(n, k, p)` qui renvoie la liste d'adjacence du graphe $G_{n,k,p}$. On utilisera la syntaxe ci-dessous pour définir un graphe vide (c'est à dire sans arc) puis on remplira progressivement les différentes listes d'adjacence à l'aide de la méthode append :

```
L = [[] for i in range(n)]
```

Question 5. Étant donné un sommet v, on appelle degré *sortant* de v le nombre d'arcs dont v est l'origine, et on dit que v est un *puit* lorsque son degré sortant est égal à 0. Parmi les graphes suivants, combien possèdent des puits ?

a) $G_{10,k,3}$ pour $0 \leqslant k < 1\,000$ b) $G_{100,k,20}$ pour $0 \leqslant k < 1\,000$

Graphe non orienté

Un graphe non orienté est un graphe orienté $G = (V, E)$ pour lequel $(i, j) \in E \iff (j, i) \in E$ (on parle dans ce cas d'arêtes plutôt que d'arcs).
La représentation d'un graphe non orienté est identique à celle d'un graphe orienté. Notons que dans le cas d'une représentation par matrice d'adjacence, cette dernière est symétrique. Il est en revanche plus difficile de caractériser un graphe non orienté en utilisant sa représentation par liste d'adjacence.

Question 6. *En utilisant la représentation par liste d'adjacence*, déterminer la plus petite valeur de k pour laquelle le graphe $G_{n,k,p}$ est non orienté lorsque :

a) $(n, p) = (5, 2)$ b) $(n, p) = (10, 10)$ c) $(n, p) = (8, 8)$

4. Parcours dans un graphe

La notion de parcours de graphe est à la base de nombreux algorithmes, par exemple pour déterminer le plus court chemin reliant deux sommets, déterminer si un graphe est connexe, tester l'existence d'un cycle, etc. Il importe de bien connaître son principe, que nous allons rappeler ici.

On se donne un sommet source $s_0 \in V$, et l'objectif d'effectuer un certain traitement sur tous les sommets accessibles à partir de s_0. L'algorithme se décrit en pseudo-code de la façon suivante :

procedure PARCOURS(sommet : s_0)
    àTraiter $\leftarrow s_0$
    déjàVus $\leftarrow s_0$
    while àTraiter $\neq \emptyset$ **do**
        àTraiter $\rightarrow s$
        traitement(s)
        for $v \in$ voisins(s) **do**
            if $v \notin$ déjàVus **then**
                àTraiter $\leftarrow v$
                déjàVus $\leftarrow v$

L'algorithme utilise deux structures de données désignées par *déjàVus* et *àTraiter* dans le pseudo-code. La première est destinée à éviter de traiter plusieurs fois le même sommet. On utilise en général une liste (ou un dictionnaire) `dejaVus` indexée par les sommets de V ; au départ on aura donc :

$\forall s \in V \setminus \{s_0\}$, `dejaVus[`$s$`] = False` et `dejaVus[`$s_0$`] = True`

La seconde contient les sommets en cours de traitement ; on utilise là encore une liste `aTraiter` ; au départ on aura :

`aTraiter = [`s_0`]`

Attention, l'ordre dans lequel on va traiter les sommets va dépendre de la façon dont va être gérée dynamiquement les entrées et les sorties de cette liste.
Quant au traitement des sommets, il dépend de ce qu'on veut obtenir : ce peut être, comme par exemple dans la question suivante, de compter le nombre de sommets accessibles à partir de s_0. Dans le pseudocode ci-dessus les sommets sont traités au moment où ils sortent de `aTraiter`, mais ce traitement pourrait aussi bien avoir lieu au moment où ils entrent dans cette liste, avec là encore un ordre de traitement qui serait modifié.

Question 7. Déterminer la valeur minimale de k pour laquelle tous les sommets de $G_{n,k,p}$ sont accessibles à partir du sommet 0 lorsque :

a) $(n,p) = (100,30)$ b) $(n,p) = (100,28)$ c) $(n,p) = (100,29)$

Parcours en profondeur et en largeur

Durant l'exécution de l'algorithme, la liste `aTraiter` supporte deux types d'opérations : ajouter et retirer un élément. On utilise par défaut les deux méthodes ci-dessous :

- `.append()` pour ajouter un élément à la file ;
- `.pop()` pour extraire un élément de la file.

La liste joue alors le rôle d'une pile : c'est le dernier élément entré qui sera le premier élément sorti.
On dit que le parcours s'effectue *en profondeur* : l'algorithme explore tous les chemins issus d'un voisin de s_0 avant de passer au traitement du voisin suivant.

Si on veut utiliser une structure de file, autrement dit lorsque les éléments sortent dans l'ordre dans lequel ils sont entrés, il faut utiliser les deux méthodes :

- `.append()` pour ajouter un élément à la file ;
- `.pop(0)` pour extraire un élément de la file.

On parle dans ce cas de parcours *en largeur* : les différents sommets sont traités par distance croissante à s_0 ; d'abord ceux qui sont à une distance 1 de s_0, puis ceux à une distance 2, etc.

Remarque. Le module `collections.deque` que vous avez peut-être utilisé l'année dernière propose des implémentations plus efficaces (en terme de performances) des piles et des files.

Question 8. Déterminer la longueur minimale d'un chemin reliant le sommet 0 au sommet $n-1$ dans le graphe $G_{n,0,p}$ (on conviendra que cette distance est égale à $+\infty$ s'il n'existe pas de tel chemin) pour :

a) $(n,p) = (100,60)$ b) $(n,p) = (100,75)$ c) $(n,p) = (1\,000,965)$

Les réponses attendues

Question 1.

a) $u_{10} \bmod 1\,000 =$ 1 *b)* $u_{500} \bmod 1\,000 =$ 764 *c)* $u_{10000} \bmod 1\,000 =$ 97

Question 3.

a) pour $(n,p) = (100,2)$: 61 *b)* pour $(n,p) = (100,10)$: 16 *c)* pour $(n,p) = (100,50)$: 6

Question 5.

a) pour $(n,p) = (10,3)$: 166 *b)* pour $(n,p) = (100,20)$: 385

Question 6.

a) pour $(n,p) = (5,2)$: 988 *b)* pour $(n,p) = (10,10)$: 428 *c)* pour $(n,p) = (8,8)$: 1886

Question 7.

a) pour $(n,p) = (100,30)$: 81 *b)* pour $(n,p) = (100,28)$: 420 *c)* pour $(n,p) = (100,29)$: 1668

Question 8.

a) pour $(n,p) = (100,60)$: 12 *b)* pour $(n,p) = (100,75)$: inf *c)* pour $(n,p) = (1000,965)$: 50