

TP - Révisions sur les graphes

Ce sujet a 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.

I - Génération pseudo-aléatoire d'une suite d'entiers

On définit une suite $(u_k)_{k \in \mathbb{N}}$ par $u_0 = 42$ et

$$\forall k \in \mathbb{N}, u_{k+1} = 1022 \times u_k \bmod 65533$$

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 $0 \leq k < 100000$.

1. Définir et remplir cette liste, puis donner les valeurs suivantes :
 - a) $u_{10} \bmod 1000$
 - b) $u_{500} \bmod 1000$
 - c) $u_{10000} \bmod 1000$

II - Représentation d'un graphe par matrice d'adjacence

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 $V = \llbracket 0, n-1 \rrbracket$, 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 $M \in M_n(\{0, 1\})$, 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 $G_{n,k,p}$ désigne le graphe de sommets $V = \llbracket 0, n-1 \rrbracket$, tel que :

$$(i, j) \in E \quad \text{si et seulement si} \quad u_{i+u_{k+j}} \text{ est divisible par } p$$

2. Définir une fonction `graphe_M(n, k, p)` qui renvoie la matrice d'adjacence du graphe $G_{n,k,p}$.
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) $(n, p) = (100, 50)$

III - 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 $n = \text{card}(V)$.

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 en $O(n + m)$ avec $m = \text{card}(E)$: une liste de listes L , où pour tout $i \in \llbracket 0, n - 1 \rrbracket$, $L[i]$ est la liste des sommets accessibles à partir de i .

4. Définir une fonction `graphe_L(n, k, p)` qui renvoie la liste d'adjacence du graphe $G_{n,k,p}$.
On utilisera `L = [[] for i in range(n)]` 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`.
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 puits lorsque son degré sortant est égal à 0.
Parmi les graphes suivants, combien possèdent des puits?
a) $G_{10,k,3}$ pour $0 \leq k < 1000$ b) $G_{100,k,20}$ pour $0 \leq k < 1000$

Grphe 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.

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) $(n, p) = (8, 8)$

IV - 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, ... 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 : s0)
  aTraiter ← s0
  dejaVus ← s0
  while aTraiter ≠ ∅ do
    aTraiter → s
    traitement(s)
    for v ∈ voisins(s) do
      if v ∉ dejaVus then
        àTraiter ← v
        déjàVus ← v
```

L'algorithme utilise deux structures de données désignées par `dejaVus` et `aTraiter` 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) indexée par les sommets de V .

Au départ on aura donc :

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

La seconde contient les sommets en cours de traitement. On utilise là encore une liste. Au départ on aura : `aTraiter = [s0]`

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.

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) $(n, p) = (1000, 29)$

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 liste;
- `pop()` pour extraire un élément de la liste.

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.

On peut aussi utiliser une structure de file : 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

L'implémentation d'une file par une liste n'est pas optimal car l'opération de "dépilage" est de complexité $O(n)$. Il existe une autre structure de données qui permet des ajouts et suppressions rapides (en $O(1)$). Il s'agit du sous-module `deque` (double ended queue) du module `collections`.

```
from collections import deque

pile = deque()
pile.append(12) # Ajout d'un élément
valeur = pile.pop() # Récupération d'un élément en fin de pile

file = deque()
file.append(12) # Ajout d'un élément
```

```
| valeur = file.popleft() # Récupération d'un élément en début de file
```

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) $(n, p) = (1000, 965)$

V - Les solutions

1. a) 1 b) 764 c) 97

2.

3. a) 61 b) 16 c) 6

4.

5. a) 166 b) 385

6. a) 988 b) 428 c) 1886

7. a) 81 b) 420 c) 1668

8. a) 12 b) inf c) 50