Exercices 36 à 39

Exercice 36 (graphes). Proposer un pseudo-code pour chacune des fonctions définies ci-dessous :

(1) La fonction **simple(G)** prend en entrée un graphe G et renvoie le booléen `vrai` si le graphe G est simple, `faux` sinon.

(2) La fonction **dmax(G)** prend en entrée un graphe G et renvoie le degré maximal obtenu parmi tous les sommets de G.

(3) La fonction **régulier(G)** prend en entrée un graphe G et renvoie le booléen `vrai` si le graphe G est régulier, `faux` sinon.

(4) La fonction **complet(G)** prend en entrée un graphe G et renvoie le booléen `vrai` si le graphe G est complet, `faux` sinon.

(5) La fonction **symétrique(G)** prend en entrée un graphe orienté G et renvoie le booléen `vrai` si le graphe G est symétrique (au sens où l'arête $A \to B$ existe ssi l'arête $B \to A$ existe), `faux` sinon.

(6) La fonction **adjacence(G)** prend en entrée un graphe orienté G et renvoie sa matrice d'adjacence.

Exercice 37 (graphes). Soit G un graphe orienté de matrice d'adjacence A (on suppose que les sommets sont numérotés de 1 à n). Trouver, pour chacune des propritétés ci-dessous, une caractérisation de la propriété au moyen de A (par exemple: G est simple ssi tous les coefficients diagonaux de A sont nuls).

(1) Le graphe G est régulier

(2) Le graphe G est complet (on suppose que G est simple)

(3) Il existe un chemin dans G reliant le sommet 1 au sommet n

(4) Le graphe G est connexe

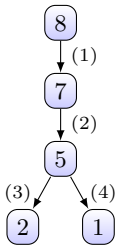
(5) Pour tout couple (x, y) de sommets de G, la distance de x à y est au plus égale à d

---

Exercice 38 (jeu à deux joueurs). Alice et Bernard décident de jouer au jeu suivant. À tour de rôle, chaque joueur reçoit un entier n, choisit un entier k selon certaines règles, et donne le nombre $n - k$ à l'autre joueur, qui continue de même. Le premier joueur qui ne peut pas jouer a perdu. Les règles imposés pour le choix de k sont les suivantes :

- le nombre k est compris entre 1 et le double du nombre choisi par l'autre joueur au tour précédent (pour Alice, qui joue en premier, le choix $k = 1$ est imposé);
- le nombre k n'a jamais été joué depuis le début de la partie;
- le résultat $n - k$ doit être positif ou nul.

On convient que Alice commence systématiquement le jeu, et que sa valeur de départ (n) est fixée d'un commun accord avec Bernard. Par exemple, si Alice commence le jeu avec $n = 8$, alors elle doit soustraire $k = 1$ (choix forcé au premier coup); Bernard reçoit donc l'entier 7, et est forcé de jouer $k = 2$ (car $k = 1$ a déjà été choisi); Alice reçoit alors l'entier 5, et peut jouer $k = 3$ ou $k = 4$. Dans les deux cas, Bernard reçoit un entier inférieur ou égal à 2, et ne peut donc plus jouer, donc Alice gagne. On peut résumer cette analyse pour $n = 8$ avec l'arbre ci-dessous, qui représente toutes les parties possibles, au nombre de 2 ici (l'étiquette de chaque nœud indique la valeur de n, et l'étiquette sur chaque arête indique le choix de k. Les feuilles correspondent aux situations perdantes (le joueur ne peut plus jouer).



(1) Dessiner l'arbre correspondant lorsque la valeur de départ est $n = 15$, puis lorsque la valeur de départ est $n = 16$ (on rapelle que c'est toujours Alice qui commence, avec cette valeur de n). Indiquer comment Bernard peut forcer le gain pour $n = 15$ (bien que ce ne soit pas lui qui commence), et comment Alice peut forcer le gain pour $n = 16$.

(2) Soit **arbre**(n, E, m) la fonction qui prend en entrée un entier $n \geqslant 1$, un ensemble d'entiers E et un entier m, et renvoie l'arbre associé à toutes les fins de parties possibles à partir de la valeur n, de l'ensemble de coup joués E (éventuellement vide) et de la contrainte $k \leqslant m$ pour le coup à venir.

Montrer que l'arbre renvoyé par **arbre**(n, E, m) a pour racine un sommet d'étiquette n et dont les enfants sont les sous-arbres retournés par **arbre**$(n-k, E \cup \{k\}, 2k)$ pour $k \in \mathbb{N}, 1 \leqslant k \leqslant n, k \leqslant m, k \notin E$.

(3) En déduire une implémentation Python de la fonction précédente sous la forme d'une fonction définie par l'en-tête

```
def arbre(n,E=set(),m=1):
```

On pourra utiliser la fonction `creer_arbre()` ci-dessous pour la création des arbres :

```
def creer_arbre(e,L=[]):
    """
    Retourne un arbre dont la racine a pour étiquette e,
    et pour enfants les éléments de la liste L (éventuellement vide)
    signature: étiquette x liste d'arbres -> arbre
    """
    return [e]+L
```

Compte-tenu des valeurs par défaut de `E` et `m`, comment interprète-t-on le résultat retourné par l'appel `arbre(n)` ?

(4) Vérifier les résultats obtenus à la question 1 en examinant les résultats retournés par `arbre(15)` `arbre(16)`.

(5) Soit A l'arbre des parties possibles pour un entier n donné. On définit une fonction booléenne g sur les nœuds de A avec la règle récursive suivante :

- si $g(y) =$ faux pour au moins un enfant y de x, alors $g(x) =$ `vrai`;
- dans tous les autres cas (et en particulier lorsque x est une feuille de A), $g(x) =$ `faux`.

Calculer la fonction g sur tous les nœuds des arbres de la question 1. Comment s'interprète la valeur que prend g pour la racine de A ?

(6) Écrire une fonction Python `g(A)` qui prend en entrée un arbre A, et renvoie la valeur de la fonction g pour la racine de A. Pour le parcours de l'arbre A, on utilisera la fonction `enfants()` ci-dessous, qui retourne la liste des enfants de la racine d'un arbre (voir cours).

```
def enfants(A):
    "retourne la liste des enfants de la racine de l'arbre A"
    return A[1:]
```

Vérifier les résultats de la question 1 en calculant `g(arbre(15))` et `g(arbre(16))`.

(7) Déterminer tous les entiers $n \leqslant 60$ pour lesquels Alice a une stratégie gagnante.

Exercice 39 (mesure de volumes).

On dispose de 2 récipients, un récipient A de 3 litres et un récipient B de 7 litres, tous deux initialement vides. Si l'on ne dispose d'aucun autre récipient ni moyen de mesurer des volumes, on peut néanmoins mesurer un volume d'un litre en procédant comme suit:

- remplir B (complètement)
- vider B dans A (jusqu'à ce que A soit plein)
- jeter le contenu de A
- vider B dans A (jusqu'à ce que A soit plein)

$\rightarrow$ il reste alors 1 litre dans B

Si l'on note sous la forme d'un couple (x, y) une configuration donnée (A contient x litres et B contient y litres), alors l'algorithme précédent peut s'écrire sous la forme d'une suite de configurations :

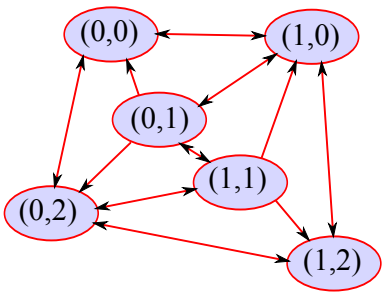
$$(0,0) - (0,7) - (3,4) - (0,4) - (3,1)$$

(1) Proposer un schéma similaire pour mesurer un volume de 2 litres.

(2) Étant donnés deux récipients A et B de contenances respectives a et b litres (avec a et b entiers et $1 \leqslant a \leqslant b$), on souhaite déterminer le graphe (simple orienté) des configurations accessibles à partir de la configuration vide. Un sommet du graphe représente une configuration, et chaque arête correspond à l'une des manipulations élémentaires suivantes :

- remplir (complètement) l'un des récipients;
- jeter le contenu d'un des récipients;
- vider (au maximum) l'un des récipients dans l'autre (à la fin, le premier est vide ou le deuxième est plein).

Par exemple, si $a = 1$ et $b = 2$, on obtient le graphe orienté ci-dessous :



Proposer le pseudo-code d'une fonction **configurations(a,b)** qui prend en entrée les deux entiers précédents a et b, et renvoie le graphe des configurations accessibles à partir de la configuration $(0, 0)$. On pourra construire ce graphe avec un parcours en largeur à partir de la configuration $(0, 0)$, c'est-à-dire en traitant dans une boucle les sommets d'un ensemble N (initialisé avec $\{(0, 0)\}$), et en construisant dans la boucle la version de l'ensemble N destinée à la boucle suivante, formée à partir des sommets rencontrés mais pas encore présents dans G.

(3) Implémenter la fonction précédente en Python (on représentera la graphe sous la forme d'un dictionnaire, comme nous l'avons vu en cours), et vérifier qu'elle retourne le bon résultat pour $a = 1$ et $b = 2$.

(4) Implémenter une fonction Python `volumes(a,b)`, qui renvoie l'ensemble des volumes entiers accessibles pour deux récipients de tailles respectives a et b. Cette fonction pourra faire appel à la fonction `configurations()` implémentée à la question 3. Examiner (après tri) le résultat produit par `volumes(a,b)` pour plusieurs valeurs de a et b, et établir une conjecture empirique sur l'expression générale de `volumes(a,b)`.

(5*) Implémenter une fonction Python `etapes(a,b)`, qui renvoie un dictionnaire donnant, pour chaque volume entier v accessible, le nombre minimal d'étapes à réaliser pour obtenir v (dans A ou B). En déduire le nombre d'étapes minimales nécessaire pour mesurer un volume de 16 litres à partir de deux récipients de contenances respectives 31 litres et 37 litres.

Indications

37.1 Comment s'interprète la quantité $\sum_{j=1}^{n} a_{ij}$?

37.3 Interpréter en termes de dénombrement la valeur de b_{1n}, où $B = (b_{ij})_{i,j}$ est la matrice définie par $B = A + A^2 + A^3 + \ldots + A^{n-1}$

37.4 Interpréter en termes de dénombrement la valeur de b_{ij}, où $B = (b_{ij})_{i,j}$ est la matrice définie par $B = A + A^2 + A^3 + \ldots + A^{n-1}$

37.5 Interpréter en termes de dénombrement la valeur de b_{ij}, où $B = (b_{ij})_{i,j}$ est la matrice définie par $B = A + A^2 + A^3 + \ldots + A^{d-1}$

38.1 Pour $n = 15$ les deux premiers coups sont forcés (Alice: $k = 1$ puis Bernard: $k = 2$), ensuite Alice a le choix entre $k = 3$ et $k = 4$. Montrer que dans les deux cas, Bernard peut trouver un coup qui mène Alice à une position perdante.

38.3 L'appel arbre(n,E,max) constuit un arbre dont la racine a pour étiquette n et pour enfants les sous-arbres obtenus par des appels récursifs de type arbre(n-k,E—k,2*k) pour les valeurs admissibles de k.

Le paramètre n représente le total disponible, l'ensemble E les entiers k déjà joués, et m est la borne maximale sur k due au coup précédent (ou imposée au départ)

38.5 Si x est un nœud de l'arbre, quel est le lien entre $g(x)$ et la possibilité de gagner à partir du nœud x ? (commencer par le cas des feuilles)

38.6 Utiliser une définition récursive de g.

39.1 Commencer comme pour un volume de 1 litre, et poursuivre ensuite les manipulations de base sans revenir en arrière jusqu'à tomber sur une configuration satisfaisante (2 litres dans l'un des récipients)

39.2 Compléter le pseudo-code suivant :

fonction configurations(a, b)
- // retourne le graphe des configurations accessibles depuis $(0, 0)$
- $G \leftarrow$ graphe vide (dictionnaire)
- $N \leftarrow$ .....
- tant que $N \neq \emptyset$
  - $M \leftarrow$ .....
  - pour tout élément (x, y) de N
    - fabriquer C, l'ensemble de toutes les configurations valides que l'on peut obtenir à partir de (x, y)
    - ajouter à M .....
    - ajouter à G la clé (x, y) associée à la valeur .....
  - $N \leftarrow M$
- retourner G

39.3 La construction de C (voir pseudo-code partiel ci-dessus) peut commencer comme ceci :

```
C = set()
if x<a and y>0: # vide B dans A
    d = min([y,a-x])
    C.add((x+d,y-d))
if y>0: # vide B
    C.add((x,0))
if y<b: # remplit B
    ......
......
```

39.4 La fonction `volumes()` récupère le dictionnaire G construit par `configurations(a,b)` et renvoie l'ensemble de tous les composantes (x et y) de tous les couples (x, y) contenus dans les clés de G.

39.5 On peut construire la fonction `etapes()` en modifiant légèrement la fonction `configurations()`. Dans la boucle sur les éléments (x, y) de N, on rajoute, si elles ne sont pas déjà présentes, les clés x et y dans le dictionnaire que l'on veut construire, avec comme valeur le numéro (commençant à 0) de tour de boucle principal (boucle "tant que $N \neq \emptyset$").