

# Algorithmique

## Contrôle n° 2

API  
EPITA

5 mars 2019 - 14 : 45

---

### Consignes (à lire) :

- ☐ Vous devez répondre sur **les feuilles de réponses prévues à cet effet**.
    - Aucune autre feuille ne sera ramassée (gardez vos brouillons pour vous).
    - Répondez dans les espaces prévus, **les réponses en dehors ne seront pas corrigées** : utilisez des brouillons !
    - Ne séparez pas les feuilles à moins de pouvoir les ré-agrafer pour les rendre.
    - Aucune réponse au crayon de papier ne sera corrigée.
  - ☐ La présentation est notée en moins, c'est à dire que vous êtes noté sur 20 et que les points de présentation (2 au maximum) sont retirés de cette note.
  - ☐ **Le code :**
    - Tout code doit être écrit dans le langage **Python** (pas de C, CAML, ALGO ou autre).
    - **Tout code Python non indenté ne sera pas corrigé.**
    - Tout ce dont vous avez besoin (classes, fonctions, méthodes) est indiqué en **annexe** !
    - Vous pouvez également écrire vos propres fonctions, dans ce cas elles doivent être documentées (on doit savoir ce qu'elles font).Dans tous les cas, la dernière fonction écrite doit être celle qui répond à la question.
  - ☐ Durée : 2h00
- 



**Exercice 1 (Espions – 4 points)**

Trois pays envoient chacun deux espions à une conférence : le pays 1 envoie les espions 1 et 2, le pays 2 envoie les espions 3 et 4 et le pays 3 les espions 5 et 6. Chaque espion doit surveiller tous les espions des autres pays, mais pas son propre collègue.

1. Représenter (dessiner) cette situation par un graphe d'ordre 6 dans lequel chaque arête entre deux sommets  $i$  et  $j$  signifie que  $i$  espionne  $j$ .
2. Ce graphe est-il complet ? Justifiez.
3. Quel est le degré de chaque sommet ? En déduire le nombre d'arêtes du graphe. Justifiez.

**Exercice 2 (Dans les profondeurs de la forêt couvrante – 4 points)**

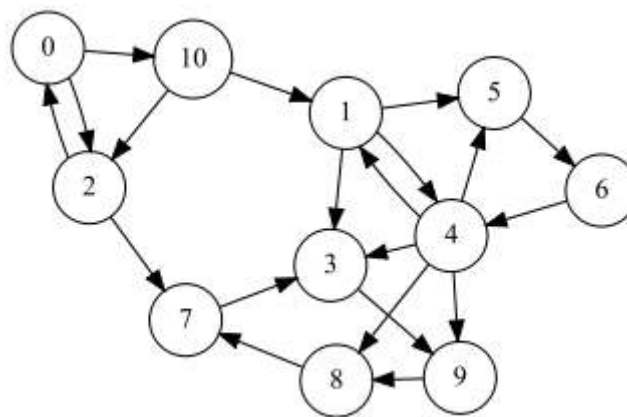


FIGURE 1 – Un graphe orienté

1. Représenter (dessiner) la forêt couvrante associée au parcours en profondeur du graphe de la figure 1. Ajouter aussi les autres arcs en les qualifiant à l'aide d'une légende explicite. On considérera le sommet 0 comme base du parcours, les sommets devant être choisis en ordre numérique croissant.
2. Remplir les vecteurs de pères et d'ordres suffixes pour chacun des sommets, correspondants au parcours de la question précédente. Les racines de la forêt seront indiquées par la valeur  $-1$  dans le vecteur de pères.

**Exercice 3 (Sérialisation – 5 points)**

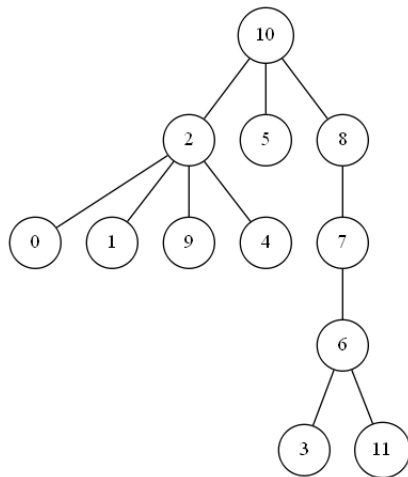


FIGURE 2 – Arbre général  $T$

Nous allons nous intéresser à une représentation alternative des arbres généraux : les vecteurs de pères. Cette représentation est linéaire et peut donc être utilisée pour stocker notre arbre dans un fichier (sérialisation).

Le principe est simple : à chaque nœud de l'arbre on associe un identifiant unique (ici la clé contenue dans chaque nœud), sous la forme d'un entier compris entre 0 et la taille de l'arbre - 1. On construit ensuite un vecteur où la case  $i$  contient l'identifiant du père du nœud d'identifiant  $i$ . La racine de l'arbre aura pour père  $-1$ .

1. Donner le vecteur de pères pour l'arbre de la figure 2.
2. Écrire la fonction `buildParentVect( $T$ ,  $n$ )` qui retourne le vecteur (représenté par une liste en Python) de pères correspondant à l'arbre  $T$  en **implémentation "premier fils - frère droit"** de taille  $n$ . Les clés de l'arbre  $T$  sont les entiers dans  $[0, n[$  (sans redondance).

**Exercice 4 (Croissants – 4 points)**

Écrire une fonction qui construit une liste des clés contenues dans un B-arbre en ordre croissant.

Exemple d'application avec B l'arbre de la figure 3 :

```

1 >>> btree2list(B)
2 [3, 5, 11, 13, 18, 25, 32, 35, 40, 44, 46, 49, 50]
    
```

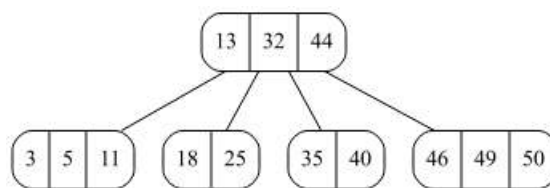


FIGURE 3 – B-Arbre B

**Exercice 5 (Mesure sur les B-arbres – 4 points)**

Dans cet exercice on se propose de mesurer la *qualité* d'un B-arbre.

Écrire la fonction `occupation(B)` qui renvoie un réel correspondant au **nombre moyen de clés par nœud** (nombre de clés / nombre de nœuds) dans le B-arbre B. La fonction retourne 0 si l'arbre est vide.

Exemple d'application avec B l'arbre de la figure 3 :

```

1 >>> occupation(B)
2 2.6
    
```

## Annexes

### Les arbres généraux

Les arbres (généraux) manipulés ici sont les mêmes qu'en td.

#### Implémentation classique

```
1 class Tree:
2     def __init__(self, key, children=None):
3         self.key = key
4         if children is not None:
5             self.children = children
6         else:
7             self.children = []
8     @property
9     def nbchildren(self):
10        return len(self.children)
```

#### Implémentation *premier fils - frère droit*

```
1 class TreeAsBin:
2     def __init__(self, key, child=None, sibling=None):
3         self.key = key
4         self.child = child
5         self.sibling = sibling
```

### B-Trees

Les B-arbres manipulés ici sont les mêmes qu'en td.

```
1 class BTree:
2     degree = None
3     def __init__(self, keys=None, children=None):
4         self.keys = keys if keys else []
5         self.children = children if children else []
6     @property
7     def nbkeys(self):
8         return len(self.keys)
```

### Fonctions et méthodes autorisées

- `len` sur les listes.
- `range`
- `append` sur les listes

Queue :

- `Queue()` retourne une nouvelle file;
- `q.enqueue(e)` enfile  $e$  dans  $q$ ;
- `q.dequeue()` supprime et retourne le premier élément de  $q$ ;
- `q.isempty()` teste si  $q$  est vide.

### Vos fonctions

Vous pouvez également écrire vos propres fonctions, dans ce cas elles doivent être documentées (on doit savoir ce qu'elles font).

Dans tous les cas, la dernière fonction écrite doit être celle qui répond à la question.