# Algorithmique
# Correction Contrôle n° 2

API – Epita

*5 mars 2019* - 14 : 45

***Solution 1*** **(Espions –** ***4 points*****)**

1. Représentation du graphe figure 1

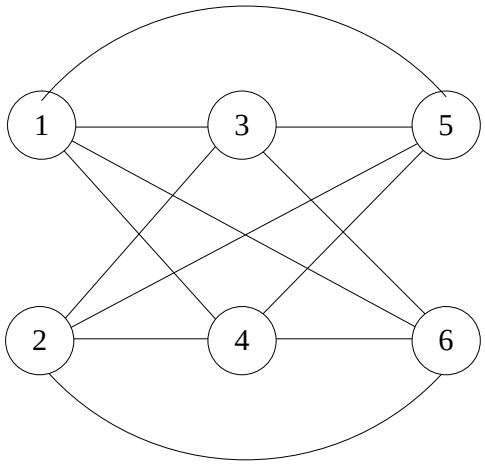


Figure 1 – Graphe représentant les liens d'espionnage.

2. Non ! Deux espions d'un même pays ne s'espionnent pas, les deux sommets correspondants (1 et 2 par exemple) ne sont donc pas adjacents.

3. Degrés des sommets : 4
   Nombre d'arêtes du graphe : 12
   Tous les sommets sont de degrés 4. Dû à la symétrie des relations, la somme des degrés est égale au double du nombre des arêtes : $(6 * 4)/2 = 12$

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

1. *Forêt couvrante et arcs supplémentaires pour le parcours en profondeur du graphe de la figure 1 du sujet :*

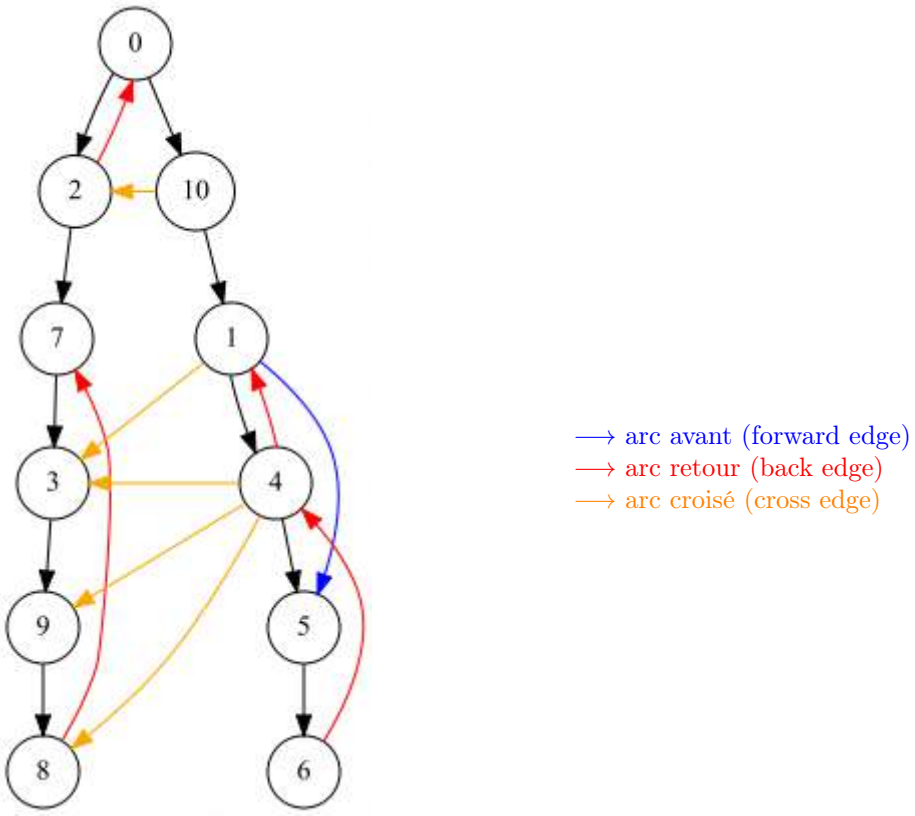


Figure 2 – DFS : Forêt couvrante

2. *Vecteurs des pères et ordres suffixes :*

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **père** | -1 | 10 | 0 | 7 | 1 | 4 | 5 | 2 | 9 | 3 | 0 |
| **suffixe** | 11 | 9 | 5 | 3 | 8 | 7 | 6 | 4 | 1 | 2 | 10 |

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

1. *Le vecteur de pères :*

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 2 | 2 | 10 | 6 | 2 | 10 | 7 | 8 | 10 | 2 | −1 | 6 |

2. **Spécifications :**
   La fonction `buildParentVect(`$T$`, `$n$`)` 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).

```
            # Recursively fills P with "parent" relations from T
            def __buildParentVect(T, P):
                C = T.child
                while C != None:
                    P[C.key] = T.key
                    __buildParentVect(C, P)
                    C = C.sibling

            def buildParentVect(T, n):
                P = [-1] * n
                __buildParentVect(T, P)
                return P
```

***Solution 4*** **(Croissants – *4 points*)**

**Spécifications :**
`BtreeToList(`$B$`)` retourne la liste des clés du B-arbre $B$ en ordre croissant.

```
                    def __BtreeToList(B, L):
                        if B.children == []:
                            for i in range(B.nbKeys):    # L += B.keys
                                L.append(B.keys[i])
                        else:
                            for i in range(B.nbKeys):
                                __BtreeToList(B.children[i], L)
                                L.append(B.keys[i])
                            __BtreeToList(B.children[B.nbKeys], L)

                    def BtreeToList(B):
                        L = []
                        if B:
                            __BtreeToList(B, L)
                        return L
```

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

**Spécifications :**
`occupation(`$B$`)` retourne le nombre moyen de clés par nœud du B-arbre $B$.

```
        def __occupation(B):          # returns the pair (nb nodes, nb keys)
            (k, n) = (B.nbkeys, 1)
            for C in B.children:
                (kc, nc) = __occupation(C)
                k += kc
                n += nc
            return (k, n)

        def occupation(B):
            if not B:
                return 0
            else:
                (k, n) = __occupation(B)
                return (k/n)
```