Algorithmique
Correction Contrôle nº 4 (C4)

Info-spé (S4) – Epita

1 mars 2017 - 9 : 30

Solution 1 **CFC** – ***4 points***

1. Voir figure 1

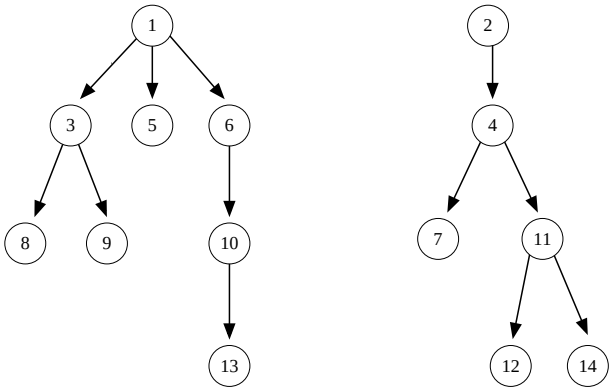
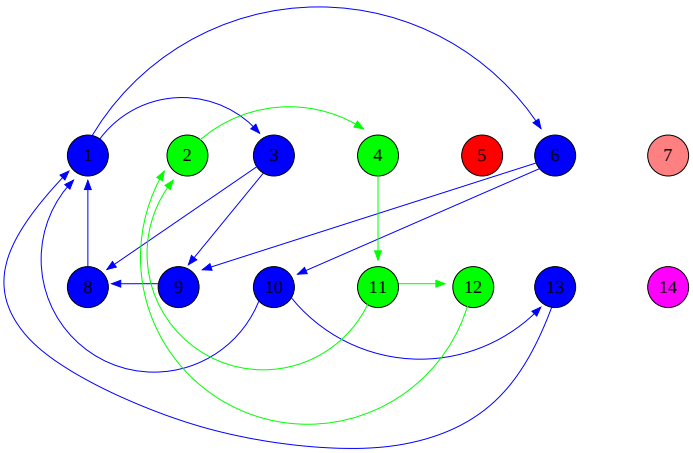


Figure 1 – Forêt couvrante associée au parcours en profondeur du graphe 1.

2. Il y a 5 composantes fortement connexes
3. Ce sont les suivantes :



Solution 2 **(Bi-connexité** – ***2 points*****)**

1. Un graphe connexe est bi-connexe si et seulement si on peut retirer un sommet sans perdre la connexité du graphe. Par conséquence, dans un graphe bi-connexe, il existe toujours deux chaînes distinctes entre toute paire de sommets.

2. **OUI**

3. Un sommet est un point d'articulation (*cut point* ou *cut vertex*) si sa suppression ajoute une composante connexe au graphe.

4. Une arrête est un isthme (*cut edge*) si sa suppression ajoute une composante connexe au graphe.

Solution 3 **(Even Tree – *5 points*)**

```
            def dfs(G, src, M):
                M[src] = True
                size = 1
                removed = 0
                for succ in G.adjLists[src]:
                    if not M[succ]:
                        (s, r) = dfs(G, succ, M)
                        removed += r
                        if s % 2 == 0:
                            removed += 1
                        else:
                            size += s
                return (size, removed)

            def even_tree(G, src = 0):
                M = [False] * G.order
                (_, removed) = dfs(G, src, M)
                return removed
```

Solution 4 **(Warshall – *3,5 points*)**

```
                def CCFromWarshall(M):
                    n = len(M)
                    cc = [0]*n
                    k = 0
                    for x in range(n):
                        if cc[x] == 0:
                            k += 1
                            cc[x] = k
                            for y in range(x+1, n):
                                cc[y] = k
                    return (cc, k)
```

Optimization :

```
                def CCFromWarshall_2(M):
                    n = len(M)
                    cc = [0]*n
                    nb = k = 0
                    x = 0
                    while nb < n:
                        if cc[x] == 0:
                            k += 1
                            cc[x] = k
                            nb += 1
                            for y in range(x+1, n):
                                if M[x][y]:
                                    cc[y] = k
                                    nb += 1
                        x += 1
                    return (cc, k)
```

Solution 5 **(Union-Find –** ***3 points*****)**

```
def CCFromEdges(L, n):
    p = build(L, n)
    cc = [None]*n
    k = 0
    for s in range(n):
        if p[s] < 0:
            k += 1
            cc[s] = k
    for s in range(n):
        cc[s] = cc[find(s, p)]
    return (cc, k)
```

Optimization :

```
def CCFromEdges_2(n, L):
    p = build(L, n)
    cc = [None]*n
    nb = k = 0
    x = 0
    while nb < n:
        if cc[x] is None:
            rx = find(x, p)
            if cc[rx] is None:
                k += 1
                cc[rx] = k
                nb += 1
            if x != rx:
                cc[x] = cc[rx]
                nb += 1
        x += 1
    return (cc, k)
```

Solution 6 **(Journey To The Moon –** ***3,5 points*****)**

```
def nbVertexInComponentsUF(p):
    nbVertex = []
    for i in range(len(p)):
        if p[i] < 0:
            nbVertex.append(-p[i])
    return nbVertex

def Moon(n, L):
    nbV = nbVertexInComponentsUF(build(L, n))
    k = len(nbV)
    ways = 0
    for a in range(k):
        for b in range(a+1, k):
            ways += nbV[a]*nbV[b]
    return ways
```