Algorithmique
Correction Contrôle n° 4 (C4)

Info-spé (S4) – Epita

28 février 2022 - 13 : 30

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

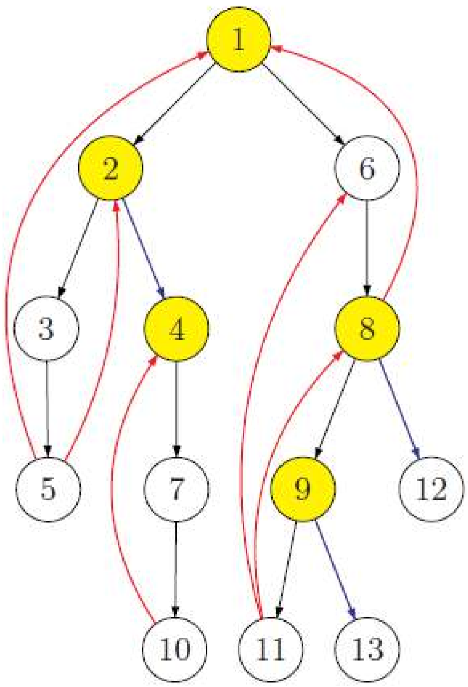

Figure 1 – Forêt couvrante.

1. Comme on peut le voir sur la figure 1, les points d'articulation sont les sommets 1, 2, 4, 8 et 9.
2. et les isthmes sont les arêtes {2,4}, {8,12} et {9,13}.
3. Enfin, les composantes biconnexes de ce graphe sont :
   — {{1,2},{1,5},{2,3},{2,5},{3,5}}
   — {{1,6},{1,8},{6,8},{6,11},{8,9},{8,11},{9,11}}
   — {2,4}
   — {8,12}
   — {9,13}
   — {{4,7},{4,10},{7,10}}

***Solution 2* (Plus Courts Chemins... – *3 points*)**

1. Dijkstra, Bellman, Johnson, Floyd, ...
2. Il existe une solution au problème de la recherche d'un plus court chemin allant de x à y **s'il existe un chemin de x à y et s'il n'y a pas de circuit absorbant**.
3. Le plus court chemin du sommet 1 au sommet 5 est : **(1,3,5)**
4. La distance (coût) de ce plus court chemin est : **0**

***Solution 3* (Warshall – *3 points*)**

Spécifications :
La fonction `CCFromWarshall(`M`)` construit la liste des composantes connexes (une liste de listes de sommets, chaque sous-liste représente une composante) du graphe G à partir de M matrice d'adjacence de la fermeture transitive de G.

```python
def CCFromWarshall(M):
    order = len(M)
    CCList = []
    n = 0
    x = 0
    inCC = [False] * order
    while n < order:
        if not inCC[x]:
            CC = [x]
            n += 1
            for y in range(x+1, order):
                if M[x][y]:
                    CC.append(y)
                    inCC[y] = True
                    n += 1
            CCList.append(CC)
        x += 1
    return CCList
```

***Solution 4* (Composante fortement connexe – *9 points*)**

1. **Level 1** : Simples parcours profondeur

   (a) *L'ensemble des sommets de la composante fortement connexe de x dans G^{-1} est S_X*

   (b) *Comment construire l'ensemble S_X en utilisant également G^{-1} ?*

   Un premier parcours de G depuis x permet de marquer les sommets accessibles depuis x.
   Un deuxième parcours de G^{-1} depuis x permet de marquer les sommets pouvant accéder à x dans G.
   S_X est l'ensemble des sommets marqués par les deux parcours.

   Level 2 : Tarjan

   (a) $S_X = \{0, 1, 8, 9\}$

   (b) *Comment ne conserver que la liste des sommets de l'arbre couvrant appartenant à S_X ?*
   Lors du parcours depuis x :
   — On conserve les sommets dans une pile (ou mieux directement une liste) en préfixe.
   — En suffixe : À chaque racine de composante r trouvée différente de x, on "supprime" (dépile) les sommets de la composante (on dépile jusqu'à la racine r).
   — Il ne reste plus que les sommets de S_X.

2. **Spécifications :**
   La fonction `component(`G`, `x`)` construit **la liste** des sommets de la composantes fortement connexe du sommet x dans le graphe orienté G.

Level 1 :

```
# return the reverse of G
def reverse_graph(G):
    G_1 = graph.Graph(G.order, True)
    for s in range(G.order):
        for adj in G.adjlists[s]:
            G_1.addedge(adj, s)
    return G_1

def __dfs(G, x, M):
    M[x] = True
    for y in G.adjlists[x]:
        if not M[y]:
            __dfs(G, y, M, mark)

# simple DFS from s: return the mark vector
def simpleDFS(G, s):
    M = [False] * G.order
    __dfs(G, s, M)
    return M


def component_naive(G, x):
    G_1 = reverse_graph(G)
    plus = simpleDFS(G, x)
    minus = simpleDFS(G_1, x)
    L = []
    for s in range(G.order):
        if plus[s] and minus[s]:
            L.append(s)
    return L
```

Level 2 :

```python
def __scc_Tarjan(G, x, pref, cpt, L):
    """
    DFS of G from x
    pref the prefix vector and cpt its counter
    L: the list of vertices in src's scc
    return (return_x, cpt) where return_x s the return value of x
    """
    cpt += 1
    pref[x] = cpt
    L.append(x)
    return_x = pref[x]
    for y in G.adjlists[x]:
        if pref[y] == None:
            (ret_y, cpt) = __scc_Tarjan(G, y, pref, cpt, L)
            return_x = min(return_x, ret_y)
        else:
            return_x = min(return_x, pref[y])

    if return_x == pref[x] and pref[x] != 1:
        # delete the vertices of the current scc from L
            y = -1
            while y != x:
                y = L.pop()
                pref[y] = G.order * 12

    return (return_x, cpt)


def component_Tarjan(G, x):
    pref = [None] * G.order
    L = []
    __scc_Tarjan(G, x, pref, 0, L)
    return L
```

***Solution 5* (Five room puzzle)**

Est-il possible de trouver un chemin qui passe une seule fois par chacune des portes ? NON

Justification :

C'est un problème de graphe eulérien : On peut associer un graphe à cette figure (en réalité un multigraphe car nous aurons des arêtes multiples) de la façon suivante : les sommets représentent les zones (y compris l'extérieur) et les arêtes représentent les passages par les portes. Le problème revient alors à trouver une chaîne Eulérienne dans ce graphe. Or, ce graphe contient 4 sommets de degrés impairs, c'est donc impossible.