Algorithmics
Correction Midterm #4 (C4)

Undergraduate 2^{nd} year (S4) – Epita

6 March 2018 - 14 : 45

Solution 1 (Cut points, cut edges – *3 points*)

1. *Spanning forest for the DFS of the graph G_1:*

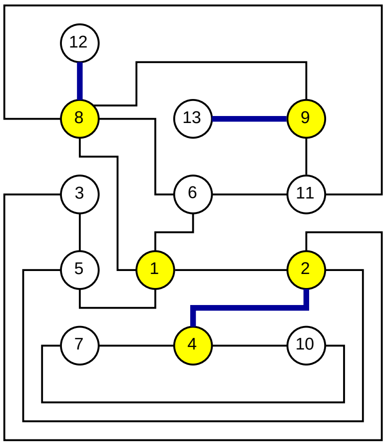


Figure 1: Graphe G_1

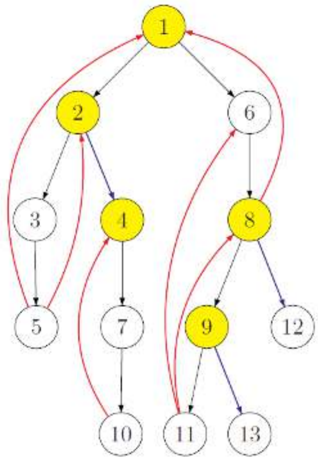


Figure 2: Forêt couvrante

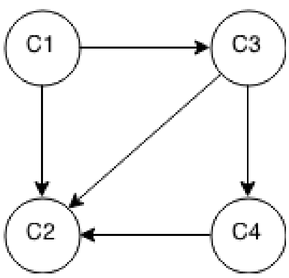
2. *Cut points of G_1:* 1, 2, 4, 8, 9
3. *Cut edges of G_1:* (2,4), (8,12), (9, 13).

Solution 2 (SCC and reduced digraphs – *5 points*)

1. Digraph → condensation
   (a) Strongly connected components of the digraph G_2:

   C_1 : {0, 1}
   C_2 : {2, 3, 4, 6, 7, 8}
   C_3 : {5}
   C_4 : {9}

   (b) Condensation of G_2:



(c) The addition of a single edge from component C_2 to the component C_1 builds a cycle that traverses all components. Thus it makes the digraph strongly connected.

2. Condensation $\rightarrow$ digraph

   (a) Vertices in the component C_2 (from 4 to 7) are unreachable from the vertex 0.

   (b) Among the following paths, which ones can not exist in G_3?

   - **3 $\rightsquigarrow$ 7**
   - 4 $\rightsquigarrow$ 21 existe
   - **18 $\rightsquigarrow$ 2**
   - **11 $\rightsquigarrow$ 15**

   (c) Adding two edges to G_3 is sufficient to meke it strongly connected.
   For instance $x_1 \rightarrow y_1$ with $x_1 \in C_6$, $y_1 \in C_2$ and $x_2 \rightarrow y_2$ with $x_2 \in C_4$, $y_2 \in C_6$.

Solution 3 **("Global Connectivity Indicators" – *6 points*)**

Global connectivity indicators measure the subdivision degree of a graph into connected components separated from each other.

1. The weighted connectivity index expresses the probability that two random vertices can be connected by a path (i.e. belong to the same connected component).

2. **Specifications:**
   The functions `indexes(G)` computes both "connectivity indexes" simple (IC_1) and weighted (IC_2) of the graph G.

```python
def __nbVertexDFS(G, s, M):
    M[s] = True
    nb = 1
    for adj in G.adjlists[s]:
        if not M[adj]:
            nb += __nbVertexDFS(G, adj, M)
    return nb

def connectivity(G):
    M = [False]*G.order
    k = 0
    IC2 = 0
    for s in range(G.order):
        if not M[s]:
            k += 1
            nb = __nbVertexDFS(G, s, M)
            IC2 += nb*nb
    IC1 = (G.order - k) / (G.order-1)
    IC2 = IC2 / (G.order * G.order)
    return (IC1, IC2)
```

***Solution 4* (Strongly Connected? – *7 points*)**

1. *Property(ies) of the first component root met:*
   The digraph is strongly connected if the first *component root* met during the traversal is the root of the unique spanning tree.

2. **Specifications:**
   The function `is_strong(`G`)` tests whether the digraph G is strongly connected.

```python
def __isStronglyConnected(G, x, pref, cpt):
    cpt += 1
    pref[x] = cpt
    return_x = pref[x]
    for y in G.adjlists[x]:
        if pref[y] == 0:
            (ret_y, cpt) = __isStronglyConnected(G, y, pref, cpt)
            if ret_y == -1:
                return (-1, cpt)
            return_x = min(return_x, ret_y)
        else:
            return_x = min(return_x, pref[y])

    if return_x == pref[x]:
        if pref[x] != 1:    # the root must be the first vertex
            return (-1, cpt)

    return (return_x, cpt)
```

```python
def isStronglyConnected(G):
    pref = [0]*G.order
    cpt = 0
    (r, cpt) = __isStronglyConnected(G, 0, pref, cpt)
    return (r != -1) and (cpt == G.order) # all vertices have been encountered
```