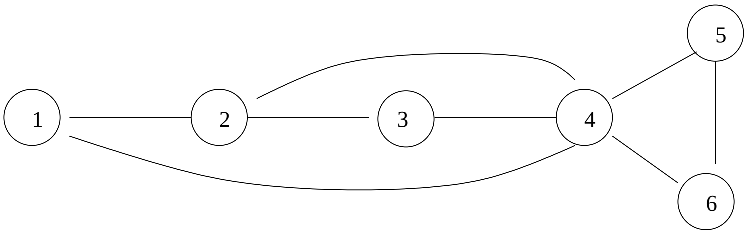
Algorithmique des Graphes – TD 4

Exercice 1

Dans cet exercice on va utiliser l'algorithme PP de parcours en profondeur pour déterminer les points d'articulation d'un graphe non-orienté. Un point d'articulation est un sommet dont la suppression augmente le nombre de composantes connexes du graphe.

1-Déterminer les points d'articulation du graphe G suivant :



On fixe un sommet `r` et on considère l'arborescence de liaison `T(r)` (resp. la fonction `d`) définie par le parcours en profondeur de G à partir de `r`. Pour tout sommet `v`, on définit `l[v]` comme la plus petite valeur de `d[u]` où `u` est un sommet accessible depuis `v` par un chemin dans `T(r)`, ou bien par un chemin dans `T(r)` suivi d'un arc de retour.
Par exemple, pour le graphe G ci-dessus on aura `l[1]=l[2]=l[3]=l[4]=1` et `l[5] = l[6] = 4` si `r=1`.
2-Modifier l'algorithme PP pour calculer `l[v]` pour tout sommet `v` d'un graphe G.
3-Etablir la caractérisation suivante :
`v` est un point d'articulation si et seulement si
- soit `v=r` et `v` a au moins deux fils dans `T(r)`,
- soit $\texttt{v} \neq \texttt{r}$ et `v` possède au moins un fils `w` dans `T(r)` tel que $\texttt{l[w]} \geqslant \texttt{d[v]}$.

4-Modifier l'algorithme PP pour calculer, pour tout sommet `v` d'un graphe, `pa[v]` qui vaut `vrai` si et seulement si `v` est un point d'articulation.
5-Tester l'algorithme modifié sur le graphe G ci-dessus et sur le graphe obtenu en ajoutant à G l'arête $(6, 3)$.

Solution exercice 1 :

Ajout de l[] : on implémente l' égalité :

$$l(u) = \min(\{l(v) \mid (u,v) \text{ arc de T(r)}\} \cup \{d(u)\} \cup \{d(v) \mid (u,v) \text{ arc de retour }\})$$

On conserve `PP(G)` mais on modifie comme suit `Visiter_PP(u)`.

```
Visiter_PP(u)
     visité[u] <- 'EN COURS'
*    l[u] <- d[u] <- temps <- temps + 1
     pour chaque v de Adj[u]
         faire si visité[v] = 'NON'
                alors pi[v] <- u
                      Visiter_PP(v)
*                     si l[v] < l[u]
*                     alors l[u] <- l[v]
*               sinon si d[v] < l[u] et pi[u] <> v    // arc retour
*                     alors l[u] <- d[v]
    visité[u] <- 'OUI'
    f[u] <- temps <- temps+1
```

Ajout de pa[]

```
PP(G)
    pour chaque sommet u de X[G]
        faire visité[u] <- 'NON'
              pi[u] <- nil
*             pa[u] <- faux
*             racine[u] <- faux
    temps <- 0
    pour chaque sommet u de X[G]
        faire si visité[u] = 'NON'
*             alors racine[u] <- vrai
*                   Visiter_PP(u)
```

```
Visiter_PP(u)
  1  visité[u] <- 'EN COURS'
  2  l[u] <- d[u] <- temps <- temps + 1
  3  pour chaque v de Adj[u]
  4      faire si visité[v] = 'NON'
 *5            alors si racine[u] et temps <> d[u]  //condition 1
 *5'                 alors pa[u] = vrai
  5''                pi[v] <- u
  6                  Visiter_PP(v)
  7                  si l[v] < l[u]
  8                  alors l[u] <- l[v]
* 8'                 sinon si l[v] >= d[u] et !racine[u]       //condition 2
* 8''                      alors pa[u] = vrai
  9            sinon si d[v] < l[u] et pi[u] <> v
 10                    alors l[u] <- d[v]
 11 visité[u] <- 'OUI'
 12 f[u] <- temps <- temps+1
```

Remarque :
si la graphe est connexe on peut se passer du tableau `racine` et donc le test en `*5` devient :

```
*5         alors si d[u] = 1 et temps > 2
```