Rapport de T.P. Noté
Orientation d'un graphe non orienté et d'un graphe fortement connexe

Loris Croce Loïc Castillo

Exercice 1

Calcul de *2-connexité* dû à Schmidt. On a `fig1` et `fig2` les graphes correspondant aux figures 1 et 2 du sujet et `art` le graphe correspondant à celui présenté dans l'article.

1. Calculer les composantes *2-connexes* d'un graphe. Ici, `show_2_connected` prend en paramètre une liste `chaines` une décomposition en chaines.

```
def show_2_connected(chaines):
    G1 = Graph(0)
    nbCycles = 0

    for chaine in chaines:
        if is_cycle(chaine):
            nbCycles = nbCycles+1
            for i in chaine:
                v0 = vertex_name(i[0], nbCycles)
                v1 = vertex_name(i[1], nbCycles)

                G1.add_vertex(v0)
                G1.add_vertex(v1)

                G1.add_edge([v0, v1])
        else:
            for i in chaine:
                v0 = vertex_name(i[0])
                v1 = vertex_name(i[1])

                G1.add_vertex(v0)
                G1.add_vertex(v1)

                G1.add_edge([v0, v1])
    G1.show()
```

   La fonction créée un graphe G' (`G1`) vide, et pour chaque arête présente dans chaque chaîne, elle ajoute les deux sommets, si non présent dans G' et ajoute ensuite une arête entre ces deux sommets. Si un sommet est présent dans une chaîne qui est un cycle et qui n'est pas le premier cycle, le sommet possède alors un nom composé de ' (exemple : $4'$ pour le 2^e sommet nommé 4) afin de la différencier du sommet présent dans le graphe composé des chemins.

2. Calculer les composantes *2-arêtes connexes* d'un graphe. Ici, `show_2_edges_connected` prend en paramètre un graphe `G` et une liste `deconnectantes`, sa liste d'arêtes deconnectantes.

```
def show_2_edge_connected(G, deconnectantes):
    G1 = G
    G1.delete_edges(deconnectantes)
    G1.show()
```

   La fonction affiche un graphe G' (`G1`), une copie du graphe G sans ses arêtes deconnectantes.

Exercice 2

La fonction `Exercice2` prend en paramètre `T`, l'arbre de parcours de l'algorithme de Schmidt et `deconnectantes` la liste des arêtes déconnectantes.

```
def Exercice2(T, deconnectantes):
    print "\nExercice 2 : orientation fortement connexe."
    if len(deconnectantes) == 0:
        print "Graphe fortement connexe : "
        T.show()
    else:
        print "arete deconnectante : " + str(deconnectantes)
```

S'il n'y pas d'arêtes déconnectantes on affichera l'arbre de parcours, sinon on affiche les arêtes déconnectantes.

Exercice 3

Hypothèse : Un graphe est fortement connexe si et seulement si chaque arc est présent dans au moins un circuit de G. Soit $G = (V, E)$ un graphe orienté et $u, v \in E$.

Preuve par l'absurde : On suppose qu'il existe un arc (u, v) qui n'est présent dans aucun circuit de G. Et que G est fortement connexe. Par définition si G est fortement connexe, il est possible d'aller de u à v et de de v à u.

Or cela n'est possible que si l'arc (u, v) appartient à au moins un circuit de G. Alors il y a contradiction et un graphe est fortement connexe si et seulement si chaque arc est présent dans au moins un circuit de G

Exercice 4

Hypothèse : Un graphe est fortement connexe si son graphe sous-jacent est 2-arête connexe.

Contre-exemple : On prend un graphe orienté fortement connexe $G = (V, E), |E| = 4$ de la forme :

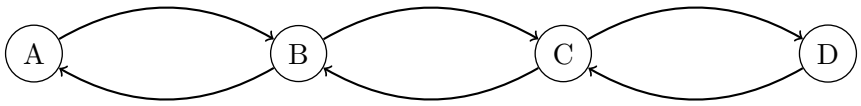

FIGURE 1 – Graphe G

Le graphe sous-jacent à G appelé G' est un chemin de taille 3. Or, chaque arête d'un chemin est une arête déconnectante. Donc G' n'est pas 2-arête connexe.

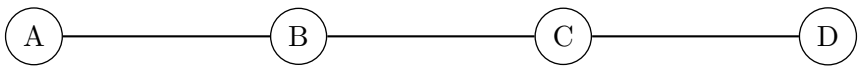

FIGURE 2 – Graphe G'

Nous pouvons donc conclure que l'hypothèse de départ est fausse.