# Algorithmique
# Correction Partiel n° 3 (P3)

INFO-SPÉ (S3) – EPITA

*19 décembre 2017* - 9 : 30

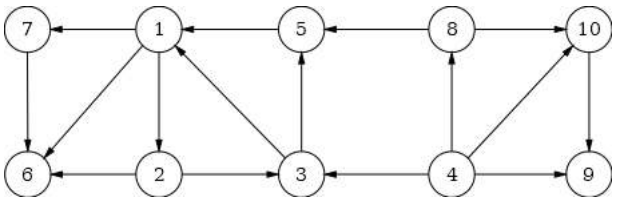
***Solution 1*** **(Union-Find –** ***3 points*****)**

1. Nombre de sommets pour chaque composante :
   $C_1$ : 4 $C_2$ : 6 $C_3$ : 4
2. Arêtes à ajouter : deux arêtes parmi $5-8$ $8-12$ $5-12$ par exemple...
3. Parmi les chaînes suivantes, celles qui ne peuvent pas exister dans $G$ :
   ☐ $3 \rightsquigarrow 7$ ☒ **$11 \rightsquigarrow 6$** ☒ **$0 \rightsquigarrow 13$** ☐ $4 \rightsquigarrow 9$

---

***Solution 2*** **(Dessiner c'est gagner –** ***4 points*****)**

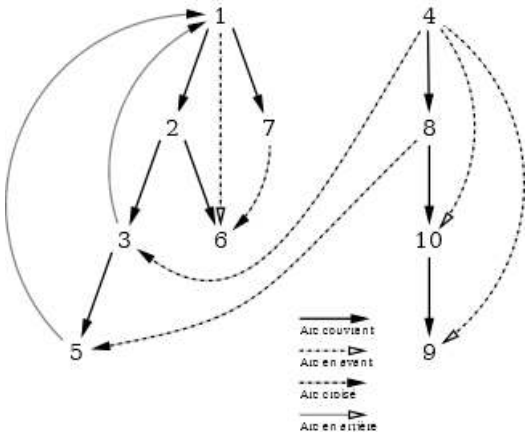
1. Le graphe orienté :




2. Le tableau des degrés est le suivant :

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|
| indegree | 2 | 1 | 2 | 0 | 2 | 3 | 1 | 1 | 2 | 2 |

3. *Forêt couvrante :*



***Solution 3*** **(Graphes bipartis (Bipartite graph) – *6 points*)**

1. $G_2$ Le premier graphe n'est pas biparti.
   $G_3$ Le deuxième est biparti avec $S_1 = \{1, 2, 5, 6, 7\}$ et $S_2 = \{0, 3, 4, 8\}$.

2. **Spécifications :**
   La fonction `bipartite(`$G$`)` indique si le graphe non orienté $G$ est biparti.

```
                    def __bipartiteBFS(G, s, Set):
                        q = queue.Queue()
                        q.enqueue(s)
                        Set[s] = 1
                        while not q.isempty():
                            s = q.dequeue()
                            for adj in G.adjlists[s]:
                                if Set[adj] == 0:
                                    Set[adj] = -Set[s]
                                    q.enqueue(adj)
                                else:
                                    if Set[adj] == Set[s]:
                                        return False
                        return True

#-------------------------------------------------------------------------------

                    def bipartite(G):
                        Set = [0] * G.order
                        for s in range(G.order):
                            if Set[s] == 0:
                                if not __bipartite(G, s, Set):
                                    return False
                        return True
```

***Solution 4*** **(Acyclic – *5 points*)**

**Spécifications :**

La fonction `is_acyclic(`$G$`)` détermine si le graphe orienté $G$ est acyclique.

```
                def __is_acyclic(G, s, M, suff):
                    M[s] = True
                    for adj in G.adjLists[s]:
                        if not M[adj]:
                            if not __is_acyclic(G, adj, M, suff):
                                return False
                        else:
                            if not suff[adj]:
                                return False
                    suff[s] = True

#-------------------------------------------------------------------------------

                    def is_acyclic(G):
                        M = [False] * G.order
                        suff = [False] * G.order
                        for s in range(G.order):
                            if not M[s]:
                                if not __is_acyclic(G, s, M, suff):
                                    return False
                        return True
```

***Solution 5*** **(What is this ? – *3 points*)**

1. Le graphe résultat ($NG$) :

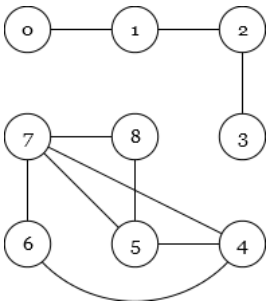



FIGURE 1 – Shuffled Graph

2. Ordre de rencontre des sommets :
   0, 8, 1, 5, 2, 3, 4, 6, 7

3. Combien de composantes connexes lorsque le graphe initial en a $k$ : $k$