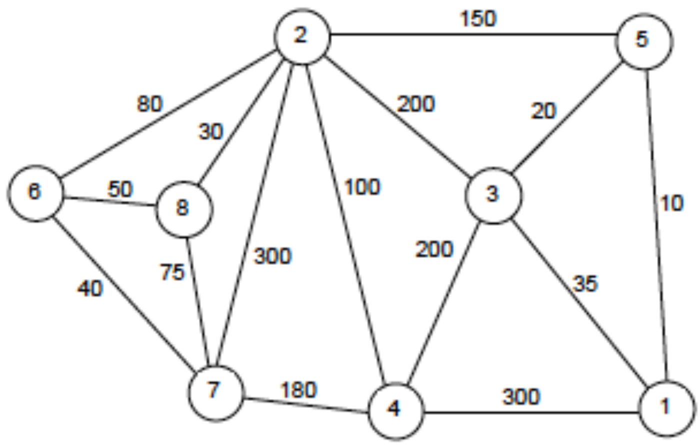
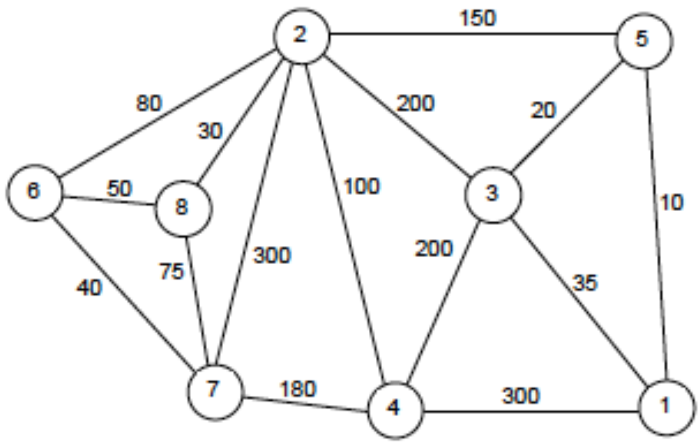
Semaine 8 – TD - Arbres Recouvrants de Poids Minimal

Exo #1. **KRUSKAL vs PRIM**

On considère le graphe non-orienté valué ci-dessous :



1. Représenter l'ARPM obtenu par l'algorithme de KRUSKAL



2. Représenter l'ARPM obtenu par l'algorithme de PRIM en prenant comme sommet source s=1.

3. Quelle propriété vérifient les deux ARPMs ainsi obtenus ? Cette propriété est-elle toujours vraie dans le cas général ? Justifier.

Exo #2. **Mise en œuvre de l'algo. de KRUSKAL en utilisant l'ADT UNION-FIND**

Rappel partie vue en CM #08. On s'intéresse au **type abstrait de données (ADT) « partition d'un ensemble »** (UNION-FIND) ou relation d'équivalence définie sur un ensemble. Il s'agit de trouver une structure de données appropriée pour gérer trois opérations :

1. Créer une relation d'équivalence « triviale » où chaque élément est seul dans sa classe.
2. Tester si deux éléments sont dans la même classe d'équivalence.
3. Faire l'union de deux classes d'équivalence pour n'en faire qu'une seule.

On notera **Sx** l'ensemble auquel appartient l'élément **x** (= la classe d'équivalence de **x**).

Ce type abstrait possède **trois primitives :**

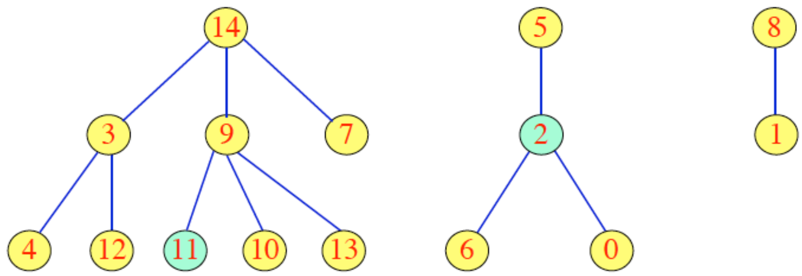
- CRÉER-ENSEMBLE(x : Elément) crée un nouvel ensemble Sx dont le seul élément (et donc le représentant) est l'élément x.

- UNION(x, y : Elément) effectue l'union des ensembles disjoints Sx et Sy. Le représentant de (Sx ∪ Sy) peut être un élément quelconque de cet ensemble. Bien souvent, on choisit, comme représentant de cette union, soit le représentant de Sx, soit le représentant de Sy.

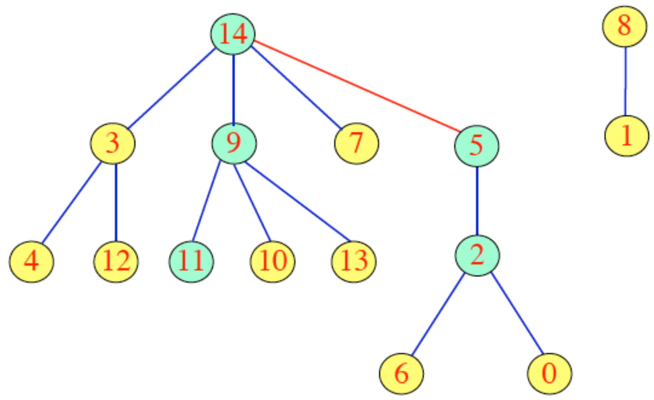
- TROUVER-ENSEMBLE(x : Elément) qui retourne le représentant de l'ensemble Sx contenant l'élément x.

Concernant l'algorithme de KRUSKAL, les ensembles sont des composantes connexes représentées par des arbres n-aires (non nécessairement binaires). Le représentant d'un arbre sera sa racine.

Exemple : soit n=15 sommets (numérotés de 0 à 14) et une forêt de trois arbres :



Le représentant du sommet #11 est le sommet #14 (racine de l'arbre contenant le sommet #11). Le représentant du sommet #2 est le sommet #5 (racine de l'arbre contenant le sommet #2).



Pour réaliser UNION(11, 2), on détermine le représentant du sommet 11, puis le représentant du sommet 2 et enfin, on joint les deux racines que sont les sommets 14 et 5 (arête en rouge).

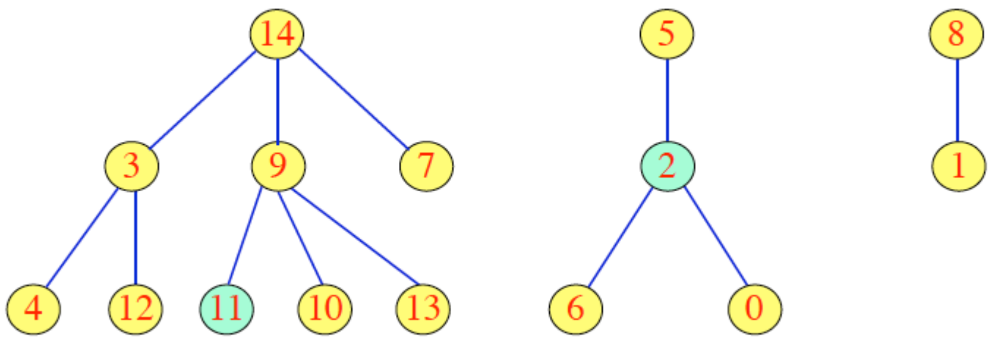
Q1. Rappeler l'algorithme de KRUSKAL en utilisant les primitives de l'adt UNION-FIND.

Soit un graphe G = (S, A, w) *et* l'algorithme de KRUSKAL défini par le pseudo-code suivant :

```
fonction KRUSKAL (G : Graphe) --> ARPM ;
        début
        T := {} ;
        pour chaque sommet v de S faire CRÉER-ENSEMBLE(v) ;
        Trier les arêtes de A par ordre croissant selon w ;
        pour chaque arête (u, v) de E (par ordre croissant selon w) faire
                si TROUVER-ENSEMBLE(u) ≠ TROUVER-ENSEMBLE(v)
                        alors   T := T U {(u, v)} ;
                                UNION(u, v) ;
        fin pour ;
        retourner T
        fin
```

Pour mettre en œuvre chacune des 3 primitives, on utilise **un tableau Pere** tel que **Pere[x] désigne le sommet père du sommet x.** Par convention, la racine de l'arbre se désigne elle-même (i.e. le sommet x est racine de l'arbre ssi x = Pere[x]).

1. Exemple : soit n=15 sommets (numérotés de 0 à 14) et une forêt de trois arbres :



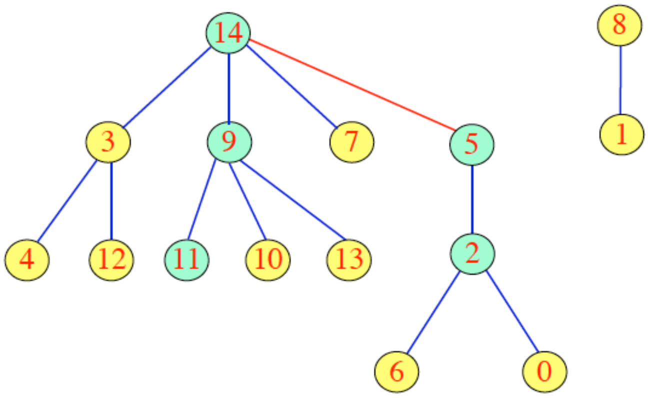
Le représentant du sommet 11 est le sommet 14 (racine de l'arbre contenant le sommet 11). Le représentant du sommet 2 est le sommet 5 (racine de l'arbre contenant le sommet 2).

Q2. Donner la représentation de chacun des 3 arbres ci-dessus à l'aide du tableau Pere.

Q3. Définir la fonction TROUVER-ENSEMBLE(x : Elément) -> Elément

Q4. Définir la procédure CRÉER-ENSEMBLE(x : Elément)

2. Exemple (suite)



Q5. Pour réaliser UNION(11, 2), on détermine le représentant du sommet 11 (sommet 14), puis le représentant du sommet 2 (sommet 5) et enfin, on joint ces deux racines (14--5 : arête en rouge).

- Définir la procédure UNION(x, y : Elément) en indiquant quel arbre devient le fils de l'autre.
- Si l'on ne prend aucune précaution particulière lors de la procédure UNION(x, y), quelle sera la hauteur maximale de l'arbre obtenu dans le pire cas ?

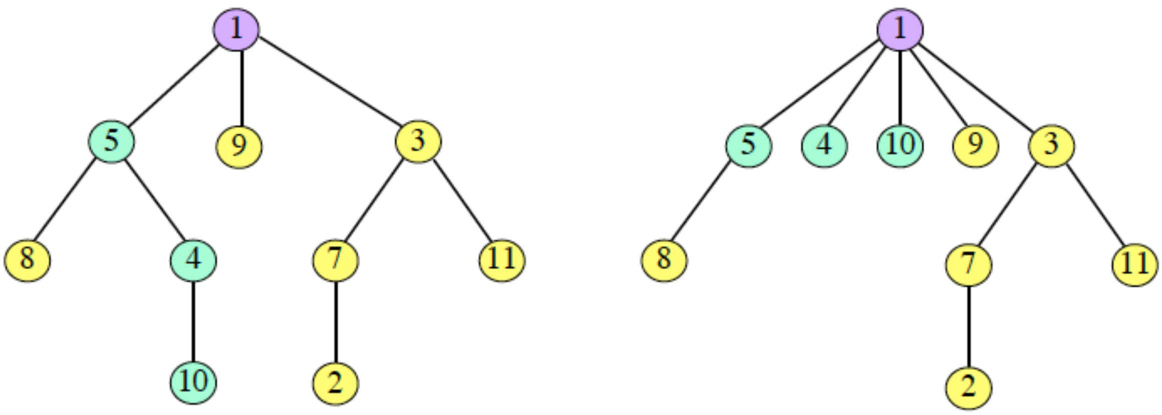
Q6. Lors de chaque union, l'arbre de plus petite hauteur devient un fils de l'arbre de plus grande hauteur. Pour cela, on associe à chaque sommet la profondeur du sous arbre dont il est la racine (via un tableau Rang). Reformuler le pseudo code des primitives CRÉER-ENSEMBLE(x) et UNION(x, y).

Q7. Montrer que si l'on fait ce choix, la procédure UNION maintient toujours la condition suivante : pour chaque arbre obtenu, on a l'inégalité $h \leq log\_2(k)$ où h représente la hauteur de l'arbre et k son nombre de nœuds.

Q8. En supposant que l'étape de tri des |A|=m arêtes selon leurs poids w s'effectue avec la meilleure complexité, quelle est la complexité temporelle (pire cas) de l'algorithme de KRUSKAL.

3. Exemple (suite et fin)

Dans la fonction TROUVER-ENSEMBLE(x), lorsque l'on remonte l'arbre pour trouver le représentant associé à x, on peut en profiter pour que chaque nœud parcouru désigne directement la racine.

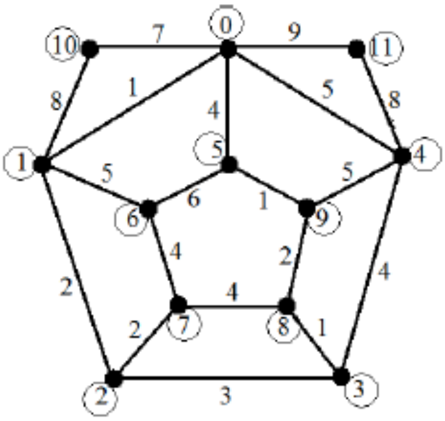


On souhaite déterminer le représentant de l'arbre contenant le nœud 10. A l'aide de la fonction TROUVER-ENSEMBLE, on remonte vers la racine (nœud #1) en visitant les sommets en vert sur l'arbre de gauche. On en profite alors pour que chaque nœud ainsi parcouru désigne son représentant (racine).

Q9. Adapter en conséquence la définition de la fonction TROUVER-ENSEMBLE(x : Elément) -> Elément

Q10. Qu'apporte la compression de chemin ? A-t-elle un impact sur la complexité pire cas ?

Exo #3. (à faire comme révision. Le corrigé est donné ci-dessous)



On considère ce graphe non orienté pondéré à 12 sommets (de 0 à 11), les poids étant indiqués sur les arêtes correspondantes.

En prenant comme point de départ le sommet 0, construire l'arbre couvrant minimal en appliquant l'algorithme de Prim.

Corrigé :