Union - Find

La structure de données UNION-FIND permet de représenter une partition d'un ensemble fini supportant les opérations suivantes :

— création d'une partition dans laquelle chaque élément est l'unique représentant de sa classe ;
— fusion des classes de deux éléments (UNION) ;
— recherche du représentant de la classe d'un élément (FIND).

Dans la suite, l'ensemble considéré est $E = \{0, 1, \ldots, n-1\}$ et une partition de cet ensemble est représenté par une forêt (*i.e.* un graphe où chaque composante connexe est un arbre). Chaque arbre de la forêt constitue une classe de la partition et la racine de chaque arbre est le représentant de sa classe.

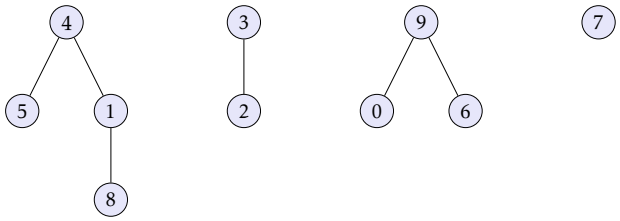

Figure 1 – Exemple d'une partition $\{\{1, 4, 5, 8\}, \{2, 3\}, \{0, 6, 9\}, \{7\}\}$.

On utilise un tableau de taille n dans lequel la case i contient le parent de i si i n'est pas la racine, et i lui-même sinon. Ainsi, l'opération FIND consiste à remonter jusqu'à la racine de l'arbre et l'opération UNION à brancher la racine d'un des deux arbres vers la racine de l'autre.

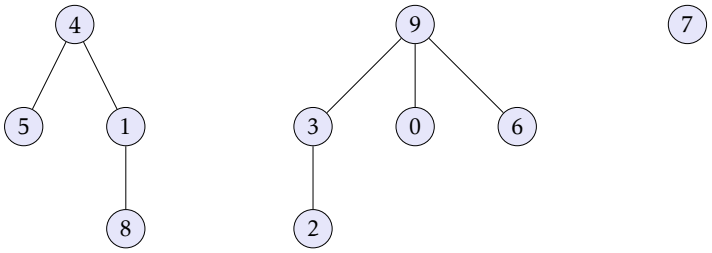

Figure 2 – Exemple d' union des classes $\{2, 3\}$ et $\{0, 6, 9\}$.

Cette organisation permet de prévoir un coût constant pour UNION et un coût en $O(h)$ pour FIND, où h désigne la hauteur de la plus grande des hauteurs des arbres de la forêt. On a alors intérêt à ce que la hauteur des différents arbres soit la plus petite possible. Pour ce faire, on utilise deux heuristiques, dites *union par rangs* et *compression de chemins*.

— La *compression de chemin* consiste, lors d'un FIND, à brancher **directement sur la racine** les différents noeuds que l'on rencontre lors d'un parcours menant du noeud initial vers la racine.
— L'*union par rang* consiste à associer à chaque noeud un *rang* qui majore la hauteur du noeud et à brancher la racine de l'arbre de moindre rang vers la racine de l'arbre de plus fort rang.

Lors de la création, le rang de chaque noeud est égal à 0 (chaque noeud est l'unique représentant de sa classe) ; par la suite, le rang sera augmenté de 1 lors de la fusion de deux arbres de même rang.

On définit alors le type suivant :

```
type partition = (int * int) array ;;
```

Chaque case i du tableau contient un couple dans lequel la première composante est le parent de i (ou lui-même si i est une racine) et la seconde composante le rang de cet élément.
On accède à chaque composante d'un couple par les fonctions `fst` (*first*) et `snd` (*second*).

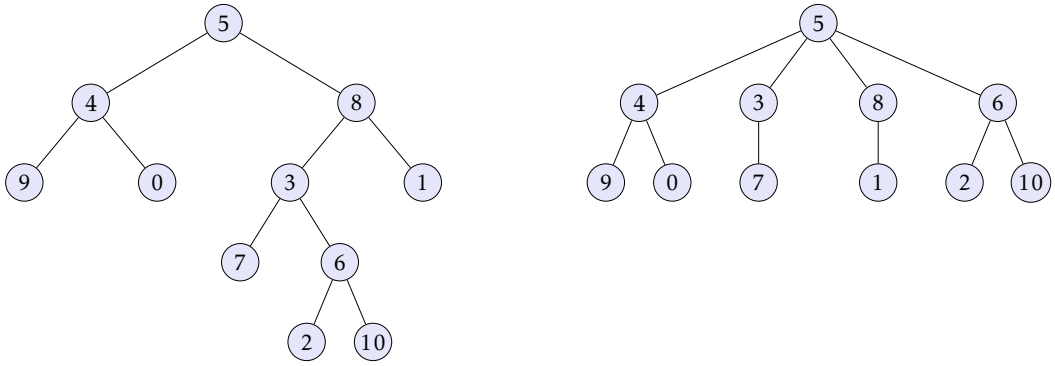

Figure 3 – Exemple (Avant/Après) de compression des chemins après avoir recherché le représentant de la classe de 6.

Question 1
Rédiger la fonction `init` qui génère une partition de $\{0, 1, \ldots, n-1\}$ en singletons.

```
init : int -> partition
```

Question 2
Rédiger la fonction `find` qui retourne le représentant de la classe de l'élément recherché en appliquant la compression du chemin parcouru (lors de cette phase, aucun **rang** n'est modifié).

```
find : partition -> int -> int
```

Question 3
Rédiger la fonction `union` qui réalise l'union par rang des classes de deux entiers passés en arguments.

```
union : partition -> int -> int -> unit
```

Remarque. Il est possible de montrer qu'avec la seule heuristique d'union par rang le coût amorti des fonctions `union` et `find` est un $O(\log(n))$. Avec l'heuristique de compression des chemins le coût de ces fonctions devient un $O(\alpha(n))$, où α est une fonction de croissance extrêmement lente (réciproque de la fonction $n \mapsto Ack(n, n)$), à un point tel que pour toutes les utilisations pratiques de cette structure on peut légitimement considérer la complexité amortie de ces deux fonctions comme étant constante.

Question 4
Application au calcul des composantes connexes d'un graphe non orienté
On considère un graphe non orienté donné par ses listes d'adjacence, autrement dit représenté par le type :

```
type voisin = int list ;;
type graphe = voisin array ;;
```

Utiliser la structure d'Union-Find pour rédiger une fonction `composantes` qui calcule les composantes connexes de ce graphe.

```
composantes : graphe -> partition
```

Question 5
On se propose de justifier expérimentalement la complexité annoncée. Pour cela, on génère uns structure de Union-Find avec $n = 1\,000\,000$ et on réalise 10^6 l'union entre classes de deux éléments pris au hasard. Vérifier expérimentalement que la hauteur maximale des arbres qui constituent la forêt ne dépasse pas 5. On pourra utiliser `Random.int` qui retourne un entier au hasard entre 0 et $n-1$.