Travaux pratiques

option informatique

Union-find

La structure de donnée appelée *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 de ce document, l'ensemble considéré est $E = \{0, 1, \ldots, n-1\}$ et une partition de cet ensemble est modélisée par une *forêt* [1], chaque arbre de cette forêt constituant une classe de la partition, la racine étant le représentant de la classe.

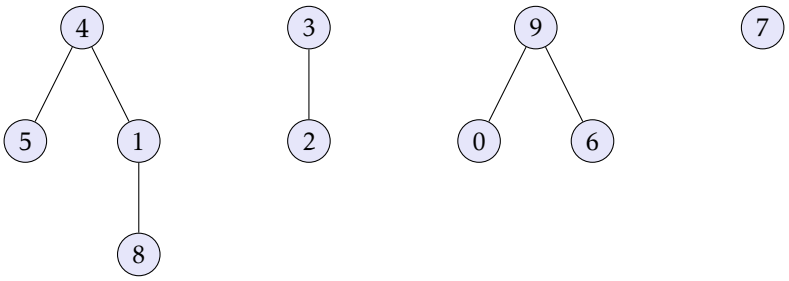

Figure 1 – Une représentation de la partition $\big\{\{1,4,5,8\},\{2,3\},\{0,6,9\},\{7\}\big\}$.

On utilise pour ce faire 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 dans le cas contraire. 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.

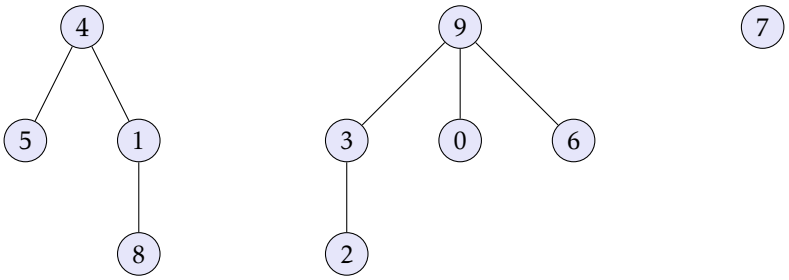

Figure 2 – Un exemple d'union des classes $\{2,3\}$ et $\{0,6,9\}$ de la partition précédente.

Cette organisation permet de prévoir un coût constant pour *union* et un coût en $O(h)$ pour *find*, h désignant la plus grande des hauteurs des différents arbres qui composent la forêt. On a donc tout intérêt à ce que la hauteur des différents arbres soit la plus petite possible. Pour ce faire, on utilise deux heuristiques, dites « union par rang » et « compression de chemin ».

- La *compression de chemin* consiste, lors d'une opération *find*, à brancher directement sur la racine les différents nœuds que l'on rencontre lors d'un parcours menant du nœud initial vers la racine (illustration figure 3).
- L'*union par rang* consiste associer à chaque nœud un *rang* qui majore la hauteur du nœud 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 nœud est égal à 0 (chaque nœud 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.

1. Rappelons qu'une forêt est un graphe dont chacune des composantes connexes est un arbre.

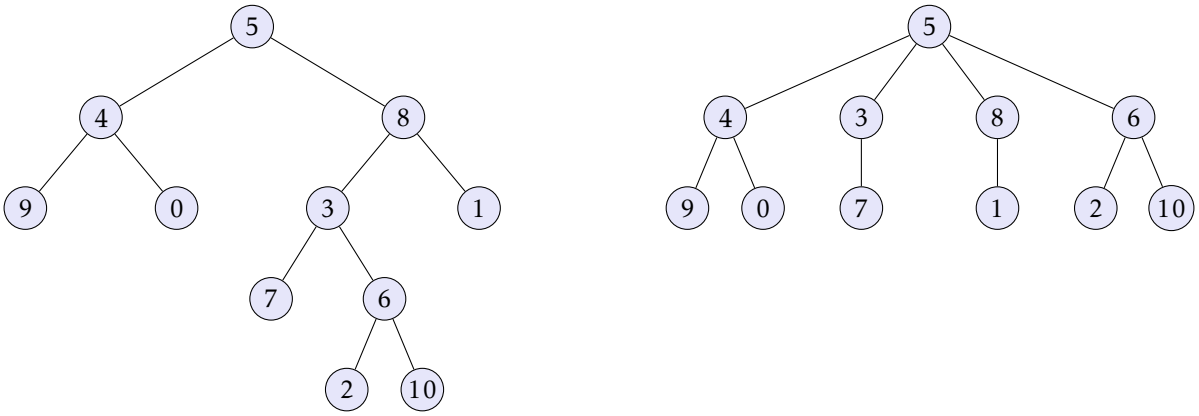

FIGURE 3 – La compression de chemin après avoir recherché le représentant de la classe de 6.

Ceci nous amène à définir le type :

```
type partition == (int * int) vect
```

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.

Question 1. Rédiger fonction **`init`** qui génère une partition de $\{0, 1, 2, \ldots, n-1\}$ 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 paramètres.

```
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 [2], 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* [3].
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 vect ;;
```

Utiliser la structure d'union-find pour rédiger une fonction qui calcule les composantes connexes de ce graphe.

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

Question 5. On se propose de justifier expérimentalement la remarque faite plus haut. Pour se faire, on génère un union-find avec $n = 1000000$ et on réalise un million de fois l'union entre les 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 la fonction **`int`** du module **`random`** : **`int n`** retourne un entier pris au hasard dans $[\![0, n-1]\!]$.

2. Pour ceux qui connaissent la fonction d'ACKERMANN, α est la fonction réciproque de $n \mapsto \text{Ack}(n, n)$.
3. L'autre grande application de la structure d'union-find en théorie de graphes est l'algorithme de KRUSKAL pour calculer un arbre couvrant de poids minimal.