

Exercice : Union-Find

La structure Union-Find sert à représenter des classes d'équivalences (ou : une partition) d'un ensemble E . Pour simplifier, on suppose $E = \{0, \dots, n-1\}$. Une structure Union-Find `uf` doit posséder 3 opérations:

- **create** `n` : crée une partition de $\{0, \dots, n-1\}$ où chaque élément est seul dans sa partition.
- **find** `uf i` : renvoie un *représentant* de la classe de `i`.
- **union** `uf i j` : réunit les classes de `i` et `j`.

On implémente chaque classe comme un arbre (non binaire à priori) dont les sommets sont les éléments de la classe et la racine son représentant.

On représente chaque arbre par un tableau `pred`, où `pred.(i)` est le père de `i` et `pred.(i) = i` ssi `i` est représentant de sa classe.

union `uf i j` a pour effet de « brancher » le représentant de `i` sur celui de `j` (le représentant de `i` devient un fils du représentant de `j`).

1. Pourquoi ne représente t-on pas ces arbres par le type persistant habituel?
2. Dessiner les arbres représentés par le tableau `pred = [17; 5; 6; 3; 4; 3; 4; 3]`.

find `uf i` doit trouver la racine de l'arbre contenant `i`, ce qui demande une complexité linéaire en la hauteur de l'arbre. Deux optimisations permettent de diminuer cette hauteur:

- *Compression de chemin*: lors d'un appel **find** `uf i`, le représentant de `i` devient le père de `i`. Ainsi un appel ultérieur à **find** `uf i` sera plus rapide.
 - *Union par rang*: associe à chaque sommet un rang (initialement 0) qui majore la hauteur de l'arbre. On branche la racine de l'arbre de moindre rang vers la racine de l'arbre de plus fort rang (en cas d'égalité, on branche arbitrairement). On stockera dans un tableau `rang` le rang de chaque sommet.
3. Écrire **create**, **find**, **union** en utilisant la compression de chemin et l'union par rang.

On peut montrer que la complexité *amortie* de **find** et **union** est $O(\alpha(n))$ où α est une fonction à croissance très lente¹: $\alpha(n) \leq 4, \forall n \leq 16^{512}$. **find** et **union** sont donc «quasiment en temps constant».

4. Comment pourrait-on adapter la structure pour pouvoir ajouter des éléments?

¹son inverse, la fonction de Ackermann, est célèbre en informatique