

7 Appartenance-union

W_(fr)

7.1 Connexité

Problème de connexité. Beaucoup d'objets, avec connexions entre eux. Question : est-ce qu'il existe une connexion entre deux objets x, y ?

Applications :

- ★ classes d'équivalence
- ★ connexité dans réseaux (ordinateurs, électronique, interaction moléculaire, société)
- ★ séquençage de génomes (objets = morceau de séquence, connexion = chevauchements de séquences)
- ★ algorithmes sur graphes (arbre couvrant, ancêtre commun)

Abstraction. On veut identifier les classes d'équivalence dans l'ensemble $\{0, 1, 2, \dots, n-1\}$, définies par une relation d'équivalence (réflexive, symétrique et transitive). On identifie chaque classe d'équivalence par un élément unique : son **élément canonique**, c'est à dire un entier de $\{0, 1, 2, \dots, n-1\}$. Opérations :

W_(fr)

- ★ **find**(x) retourne l'élément canonique de la classe de x («appartenance») : **find**(x) = **find**(y) si et seulement si les deux appartiennent à la même classe (=connexes)
- ★ **union**(x, y) établit l'équivalence (connexion) entre x et y
- ★ **makeset**(x) forme une classe avec le seul élément x . Au début, chaque élément forme une classe d'équivalence en soi : on initialise la structure par **for** $x \leftarrow 0, 1, \dots, n-1$ **do** **makeset**(x).

Exemple :

`union(3, 4); union(4, 9); union(8, 0); union(2, 3); union(7, 4); union(6, 4); union(5, 0); find(2); find(6)` (7.1)

7.2 Solution naïve

On stocke la classe de chaque élément dans un tableau `id[0..n-1]`. En conséquence, la mise à jour prend $\Theta(n)$ temps dans le code d'**union**.

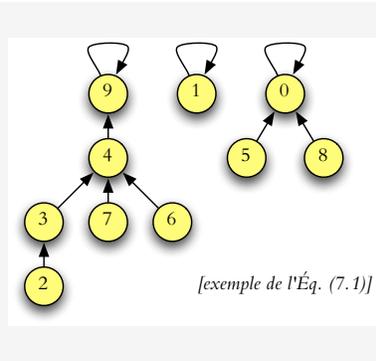
```

MAKESET
  id[x] ← x
FIND(x)
  return id[x]
UNION(x, y)
  p ← FIND(x); q ← FIND(y) // déterminer la classe à travers l'interface
  if p ≠ q then // il faut placer tous les membres de classe p dans classe q
    for z ← 0, 1, ..., n-1 do if id[z] = p then id[z] ← q

```

7.3 Solution avec un forêt d'arbres

Dans la deuxième solution, on représente chaque classe par un arbre enraciné à l'élément canonique. Il suffit de stocker `parent[0..n-1]`; on met `parent[x] = x` à la racine. Ainsi, on peut lier les éléments canoniques de deux classes en $\Theta(1)$, et `find(x)` prend un temps proportionnel à la profondeur de x .



```

MAKESET(x)
  parent[x] ← x
FIND(x) // chercher l'élément canonique de la classe de x
  while x ≠ parent[x] do x ← parent[x]
  return x
JOIN(p, q) // fusionner deux classes distinctes avec éléments canoniques p ≠ q
  parent[p] ← q
UNION(x, y)
  U1 p ← FIND(x); q ← FIND(y)
  U2 if p ≠ q then JOIN(p, q)

```

7.4 Union par rang

Afin de contrôler la hauteur des arbres et ainsi assurer l'exécution rapide de find, on introduit un autre tableau $\text{rang}[0..n-1]$. À tout temps, $\text{rang}[x]$ est une borne supérieure sur la hauteur du sous-arbre de x .

```

MAKESET(x)
M1 parent[x] ← x; rang[x] ← 0
JOIN(p, q)
J1 if rang[q] < rang[p] then échanger p ↔ q // assurer rang[p] ≤ rang[q]
J2 if rang[p] = rang[q] then rang[q] ← rang[q] + 1
J3 parent[p] ← q

```

Lemme 7.1. Pour tout x avec $\text{parent}[x] \neq x$, $\text{rang}[x] < \text{rang}[\text{parent}[x]]$.

Lemme 7.2. Le nombre d'éléments dans un arbre enraciné à x est supérieur ou égal à $2^{\text{rang}[x]}$, après toute séquence d'opérations union et find.

Démonstration. Soit $n(x)$ le nombre d'éléments de l'arbre enraciné à x . La démonstration se fait par induction dans le nombre d'opérations d'union $m = 0, 1, \dots$

Cas de base. À $m = 0$, on a $\text{rang}[x] = 0$ et $n(x) = 1$ pour tout x .

Hypothèse d'induction. On suppose que $n(x) \geq 2^{\text{rang}[x]}$ vaut pour chaque x après $m \geq 0$ opérations d'union.

Cas inductif. Soit $\text{UNION}(x, y)$ le $(m+1)$ -ème appel à UNION . Si $p = q$ dans Ligne U2, rien ne change, donc la propriété reste vraie par l'hypothèse d'induction. Autrement, on appelle $\text{JOIN}(p, q)$. Si $\text{rang}[p] < \text{rang}[q]$ dans Ligne J2, $\text{rang}[q]$ ne change pas mais son sous-arbre grandit, donc après on a $n(q) > 2^{\text{rang}[q]}$. Si $\text{rang}[p] = \text{rang}[q]$, on a (par l'hypothèse d'induction appliqué à p, q) que $n(p) + n(q) \geq 2^{\text{rang}[p]} + 2^{\text{rang}[q]} = 2^{1+\text{rang}[q]}$.

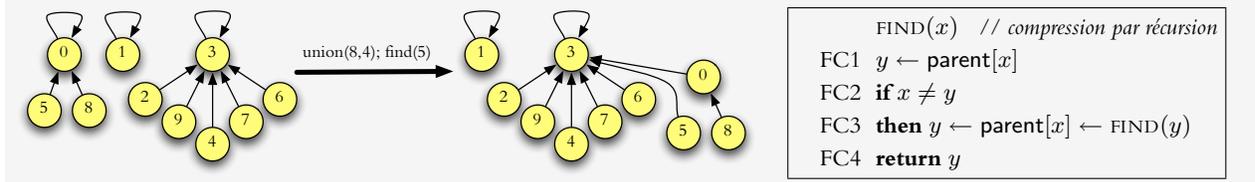
En conclusion, le lemme est valide pour toute séquence avec $m = 0, 1, 2, \dots$ opérations d'union. ■

Théorème 7.3. Une opération de $\text{find}(x)$ prend $\Theta(\log n)$ temps avec la heuristique d'union par rang.

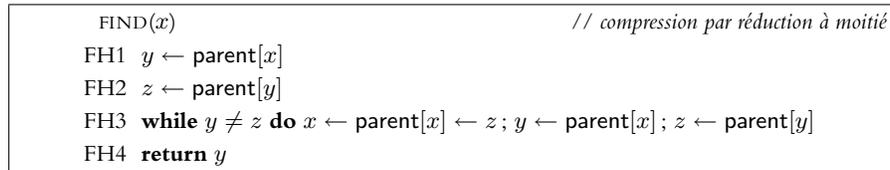
Démonstration. Par Lemme 7.1, le rang croît à chaque itération, et $0 \leq \text{rang}[p] \leq \lg n$ pour tout p par Lemme 7.2. ■

7.5 Compression de chemin

On peut réduire la profondeur aussi lors d'un find par la technique de **compression de chemin**.

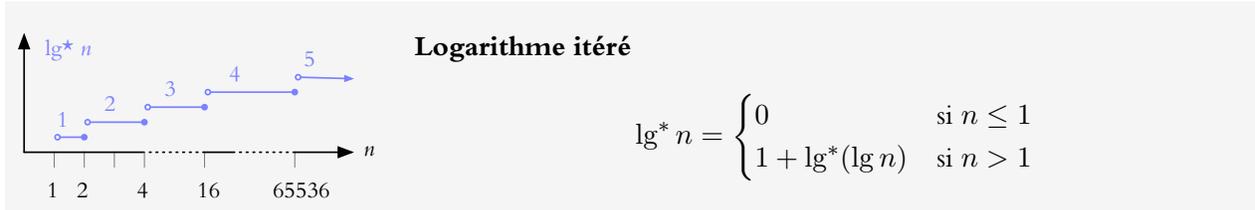


On peut éviter la récursivité avec l'astuce de **compression par réduction à moitié** (*path halving*). Cela nécessite un seul passage.



7.6 Temps de calcul

Si on implante union par rang, le temps de calcul de find est $O(\log n)$ au pire sans ou avec compression de chemin. Mais la compression mène à un coût amorti presque constant : $O(\log^* n)$ par opération, où \log^* dénote le logarithme itéré.



C'est une fonction à croissance très lente (mais monotone) : $\lg^* n \leq 5$ pour tout $n \leq 2^{65536}$.

W_(en)

Théorème 7.4. En utilisant union-par-rang et compression de chemin (ou compression par réduction à moitié), une séquence de m opérations sur n éléments prend $O(m \lg^* n)$ temps, où \lg^* dénote le logarithme itéré.

W_(fr)

REMARQUE. On peut trouver une borne plus serrée que celle du Théorème 7.4 ; avec la réciproque de la fonction d'Ackermann.

Définition 7.1 (définition de R. E. Tarjan). La fonction Ackermann $A(i, j)$ avec $i, j \geq 1$ est définie par

$$A(i, j) = \begin{cases} 2^j & \text{si } i = 1; \\ A(i - 1, 2) & \text{si } j = 1 \text{ et } i \geq 2 \\ A(i - 1, A(i, j - 1)) & \text{si } i, j \geq 2 \end{cases}$$

On définit la fonction Ackermann inverse ($m \geq n \geq 1$) par

$$\alpha(m, n) = \min\{i : A(i, \lfloor m/n \rfloor) \geq \lg n\}.$$

Ackermann inverse est une fonction à croissance même plus lente que \lg^* : $\alpha(m, n) \leq 3$ pour tout $n < 65536$ et $\alpha(m, n) \leq 4$ pour tout $n < 2^{2^{\dots^2}}$ (exponentiation 16 fois).

Théorème 7.5. En utilisant union-par-rang et compression de chemin (ou compression par réduction à moitié), une séquence de m opérations sur n éléments prend $O(m\alpha(m, n))$ temps.