UNION FIND

On dispose d'**ensembles disjoints** sur **n éléments**, on veut:

- *Tester si 2 éléments x,y appartiennent au même ensemble* : ? FIND(x)=FIND(y) où FIND(x) est l'ensemble de x
- *Fusionner 2 ensembles disjoints* contenant x et y UNION(x,y)

Une première structure naïve

Les ensembles dont indicés de 1 à n. Chaque élément connaît l'indice de son ensemble et au début x.indice=identifiant(x) <=n.

Question 1 : Quelle est la complexité du test FIND(x)=FIND(y) ?

Question 2 : Quelle est la complexité de UNION(x,y) dans le pire cas?

Question 3 : Quelle est la complexité totale si, en partant de n ensembles de taille unitaire, on itère k opérations d'UNION.

Vers une version efficace

Voir UNION-FIND issu de https://fr.wikipedia.org/wiki/Union-find

On considère maintenant une structure de données dans laquelle chaque classe est représentée par un arbre dans lequel chaque nœud contient une référence vers son nœud père. Une telle structure de forêt a été introduite par Bernard A. Galler et Michael J. Fisher en 1964, bien que son analyse détaillée ait attendu plusieurs années.

Principe

Dans une telle forêt, le représentant de chaque classe est la racine de l'arbre correspondant. **Find** se contente de suivre les liens vers les nœuds pères jusqu'à atteindre la racine. **Union** réunit deux arbres en attachant la racine de l'un à la racine de l'autre. Une manière d'écrire ces opérations est la suivante :

```
fonction MakeSet(x)
     x.parent ←  null

 fonction Find(x)
     si x.parent == null
        retourner x
     retourner Find(x.parent)

 fonction Union(x, y)
```

```
    xRacine ←  Find(x)
    yRacine ←  Find(y)
    si xRacine <>  yRacine
           xRacine.parent ←  yRacine
```

Question 4 : Dessiner les structures obtenues par la séquence:

- UNION(b,a)
- UNION(d,a)
- UNION(c,b)
- UNION(p,q)
- UNION(r,q)
- UNION(c,r)

Question 5: Quel est la complexité de la séquence UNION(1,2), UNION(1,3),...,UNION(1,n)

Modifions la version précédente en reliant vers la racine la plus « lourde ». Le poids d'un arbre est représenté par une notion de rang.

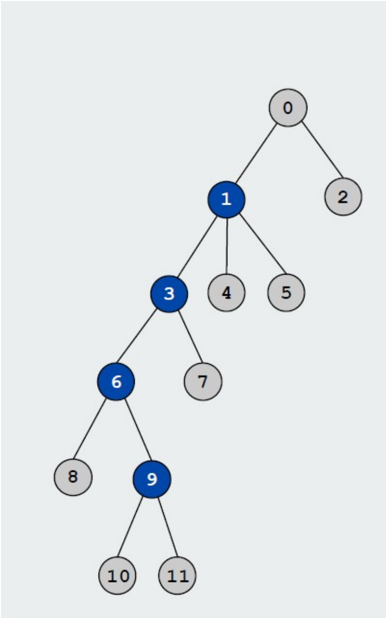
```
fonction MakeSet(x)
    x.parent ←  x
    x.rang   ←  0

 fonction UnionByrank(x, y)
    xRacine ←  Find(x)
    yRacine ←  Find(y)
    si xRacine <> yRacine
          si xRacine.rang < yRacine.rang
                xRacine.parent ←  yRacine
          sinon
                yRacine.parent ←  xRacine
                si xRacine.rang = yRacine.rang
                        xRacine.rang ←  xRacine.rang + 1
```

Question 6: Si p est père de q, quelle relation y a t'il entre le rang de p et celui de q ?

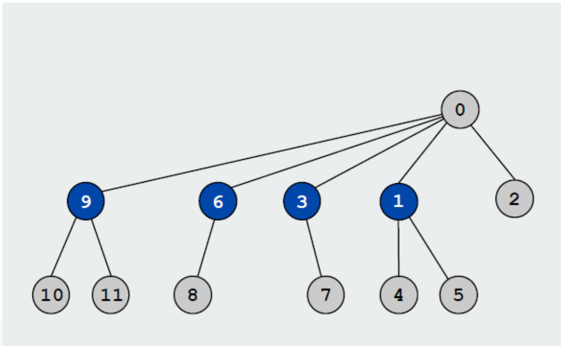
Question 7: si la racine d'un sommet x a changé k fois, combien d'éléments y a t'il au moins dans l'ensemble de x ?

Question 8: Supposons qu'une séquence de m opérations comportant m MakeSet et n UNION ou FIND. Quelle est la complexité de UNION-FIND en utilisant UnionByRank ?

Notons qu'on peut encore améliorer la structure avec une opération simple de *compressions de chemins* pour diminuer encore la profondeur de l'arbre.



→ FIND(9) →



La complexité tombe à O(n $\alpha(n)$¿ où $\alpha(n)$ est la fonction inverse d'Ackermann qui est une fonction croissante mais d'une croissance ridicule. En pratique, on considère que $\alpha(n) < 5$ même pour n > 10^100.

On peut également montrer que la complexité est O(n log*(n)) où log*(n) désigne le logarithme itéré de n, c'est-à-dire le plus petit nombre d'itération de logarithme jusqu'à obtenir log(log(log(…(log n)))) < 1.

Exercice 9. Construire un exemple où le rang atteint 2 si on utilise la compression de chemins.