Ensembles disjoints

Plan

I. Formulation initiale
II. Définitions sur les ensembles disjoints
III. Ensembles disjoints sous forme d'arbres
IV. Compression de chemin
V. Algorithme de Kruskal

Plan

I. Formulation initiale
II. Définitions sur les ensembles disjoints
III. Ensembles disjoints sous forme d'arbres
IV. Compression de chemin
V. Algorithme de Kruskal

I – Formulation initiale

Problématique:

- Supposons que l'on veuille séparer un ensemble d'éléments en sous-ensembles disjoints et de satisfaire un critère de séparabilité.

Exemples:

- Former deux équipes de football de force égale (quartiers).
- Séparer des *process* sur deux ou plusieurs processeurs sans surcharger aucun processeur.
- Séparer un ensemble de valeurs en deux ou plusieurs sous-ensembles disjoints égaux au mieux.

I – Formulation initiale

Cas (trop) simple :

- On dispose des nombres:

  2, 10, 3, 8, 5, 7, 9, 5, 3, 2

- On veut former deux ensembles égaux

I – Formulation initiale

Approche initiale:

- On répartit les nombres de valeur égale sur les deux ensembles:
  {2, 3, 5} ; {2,3, 5} / 10 8 7 9

- On trouve une solution pour les nombres restants:
  {10, 7} ; {8, 9}

- On unit les nouveaux ensembles avec les précédents
  {2, 3, 5, 7, 10} ; {2, 3, 5, 8, 9}

I – Formulation initiale

Cas typique:

- L'approche initiale n'est pas toujours utilisable.
- Supposons que nous ayons eu:

  771, 121, 281, 854, 885, 734, 486, 1003, 83, 62

- Que faire? Sachant qu'il existe un partitionnement parfait!

I – Formulation initiale

Cas typique:

- Il est possible de représenter chaque combinaison de nombres par son équivalent binaire (on a deux ensembles):

1	0	1	0	0	1	0	1	1	0
771	121	281	854	885	734	486	1003	83	62

Il suffit alors de parcourir toutes les combinaisons possibles pour trouver celles qui satisfont la condition. C'est là une approche de force brute.

I – Formulation initiale

Cas typique:

- Dans notre cas, elle permet de trouver la solution assez rapidement:

1	0	1	1	0	1	0	0	0	0
771	121	281	854	885	734	486	1003	83	62

711 + 281 +854 +734 = 2640
121 + 885 + 486 + 1003 + 83 +62 = 2640

I – Formulation initiale

Cas typique:

- Cette approche possède une complexité exponentielle $O(2^n)$:

1	0	1	1	0	1	0	0	0	0
771	121	281	854	885	734	486	1003	83	62

Peut-on faire mieux?

Que dirait-on d'un problème où il faudrait répartir l'ensemble de départ en plusieurs sous-ensembles disjoints?

Plan

I. Formulation initiale
II. Définitions sur les ensembles disjoints
III. Ensembles disjoints sous forme d'arbres
IV. Compression de chemin
V. Algorithme de Kruskal

II – Définitions sur les ensembles disjoints

Définitions:

Soit une fonction de relation R sur un ensemble S. Pour toute paire d'éléments (a,b) de S, on dit que a et b sont en relation si R(a, b) renvoie vrai.

Par exemple, dans le partitionnement égal de l'exemple précédent:

1	0	1	1	0	1	0	0	0	0
771	121	281	854	885	734	486	1003	83	62

On dira que a=771 et b=281 sont en relation si on définit {R(a,b) = vrai }ssi{a et b font partie de la même partition}

II – Définitions sur les ensembles disjoints

Définitions:

On dit de R qu'elle est une relation d'équivalence si

1. R est réflexive : R(a, a) est vrai pour tout a dans S
2. R est symétrique : R(a, b) ssi R(b, a)
3. R est transitive : {R(a, b) ET R(b, c)} ⇒ {R(a, c)}

Par exemple, la relation R(a, b) = a ≤ b n'est pas une relation d'équivalence. (Pourquoi?)

Par exemple, la relation R(a, b) = a et b vont au même cours n'est pas une relation d'équivalence. (Pourquoi?)

II – Définitions sur les ensembles disjoints

Définitions:

Un ensemble d'équivalents d'un élément a de S est le sous-ensemble de S comprenant tous les éléments x de S tels E(a, x) est vrai.

Pour une relation d'équivalence E sur S, le problème est souvent de répondre si E(a, b) est vrai...

E(1003, 62) est-il vrai?

II – Définitions sur les ensembles disjoints

Définitions:

Une ensemble d'équivalents d'un éléments a de S est le sous-ensemble de S comprenant tous les éléments x de S tels que E(a, x) est vraie.

Pour les besoins d'identification, on associe à chacun des ensembles d'équivalence (qui sont par définition disjoints) un élément représentatif.

1	0	1	1	0	1	0	0	0	0
771	121	281	854	885	734	486	1003	83	62

Par exemple, l'ensemble {771, 281, 854, 734} peut être représenté par un de ses éléments: 771.

II – Définitions sur les ensembles disjoints

Définitions:

Pour construire les ensembles disjoints, on recourt à trois méthodes:

1. Create(a):

   Crée l'ensemble disjoint ne contenant que **a**

2. Union(a, b):

   Consiste à unir les deux ensembles disjoints auxquels appartiennent **a** et **b**

2. Find(a):

   Renvoie le représentant de l'ensemble disjoint auquel appartient **a**

II – Définitions sur les ensembles disjoints

Étape 1:

- Créer tous les ensembles disjoints.

{771}, {121}, {281}, {854}, {885},
{734}, {486}, {1003}, {83}, {62}

II – Définitions sur les ensembles disjoints

Étape 2:

- Unir les ensembles respectant la relation d'équivalence.

1. {771, 281}, {121}, {854}, {885}, {734}, {486},
{1003}, {83}, {62}

2. {771, 281,854}, {121}, {885}, {734}, {486},
{1003}, {83}, {62}

II – Définitions sur les ensembles disjoints

Étape 2:

- Unir les ensembles respectant la relation d'équivalence.

3. {771, 281, 854, 734}, {121}, {885}, {486}, {1003}, {83}, {62}

4. {771, 281, 854, 734}, {121, 885}, {486}, {1003}, {83}, {62}

5. {771, 281, 854, 734}, {121, 885, 486}, {1003}, {83}, {62}

6. {771, 281, 854, 734}, {121, 885, 486, 1003}, {83}, {62}

7. {771, 281, 854, 734}, {121, 885, 486, 1003, 83}, {62}

8. {771, 281, 854, 734}, {121, 885, 486, 1003, 83, 62}

II – Définitions sur les ensembles disjoints

Étapes subséquentes:

- Il est dès lors possible de répondre aux requêtes de Recherche.

{771, 281, 854, 734}, {121, 885, 486, 1003, 83, 62}

On supposant que le premier élément identifie l'ensemble:

Recherche(83) => 121
Recherche(734) => 771

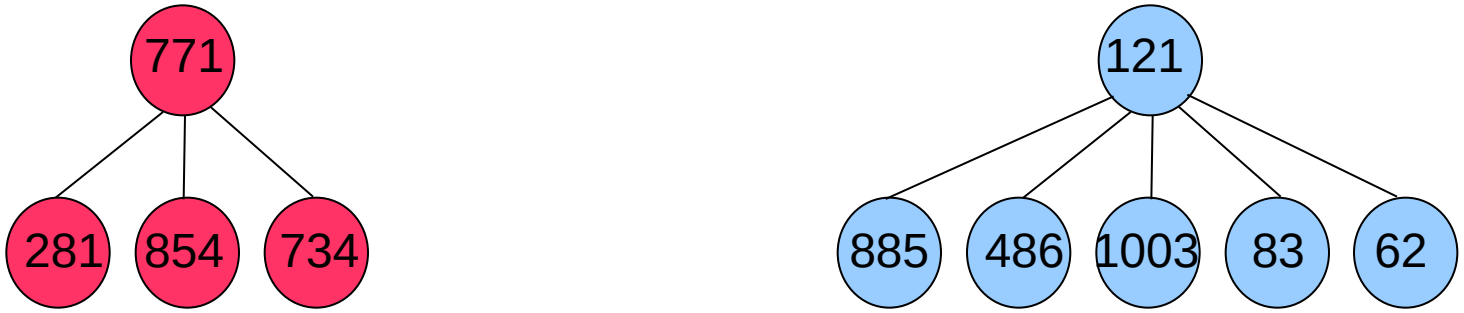
Plan

I. Formulation initiale
II. Définitions sur les ensembles disjoints
III. Ensembles disjoints sous forme d'arbres
IV. Compression de chemin
V. Algorithme de Kruskal

III – Ensembles disjoints sous forme d'arbres

Implémentation:

Il est possible d'implémenter des ensembles disjoints par des arbres:

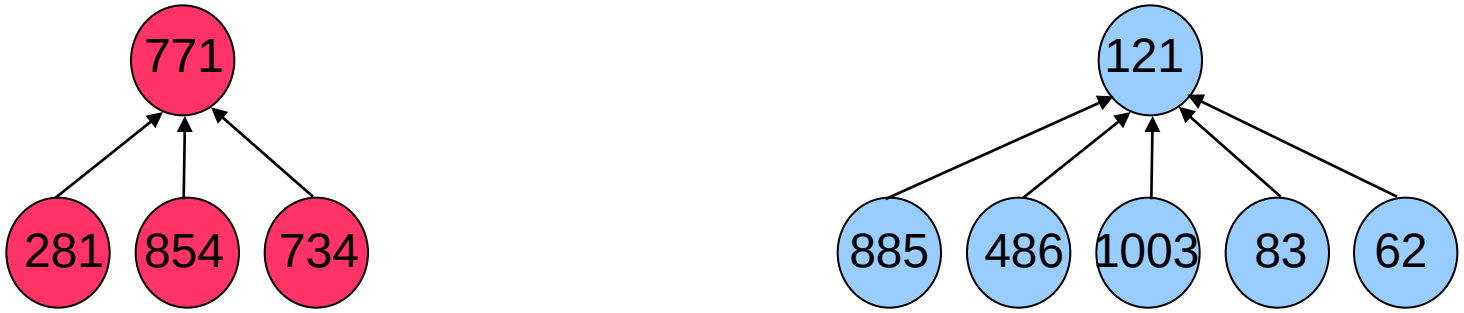


La requête Recherche(a) consiste alors à retourner la racine de l'arbre auquel appartient l'élément **a** en admettant qu'elle représente son ensemble.

III – Ensembles disjoints sous forme d'arbres

Implémentation:

Pour que la requête Find(a) puisse retourner son parent, les arcs sont orientés vers le haut:

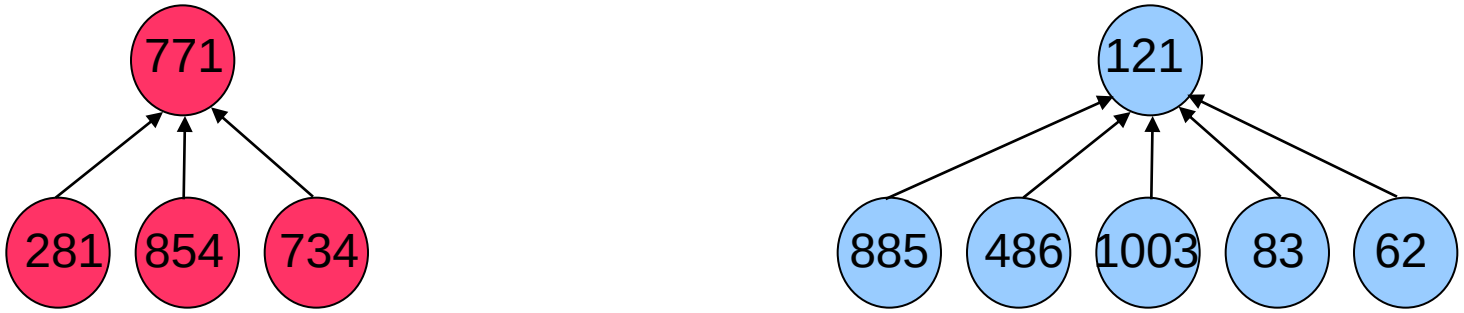


On suppose que l'on a un accès direct à chacun des nœuds! !

III – Ensembles disjoints sous forme d'arbres

Implémentation:

En admettant que l'on ait un accès direct à chacun des nœuds, les arbres peuvent être implémentés par un tableau:

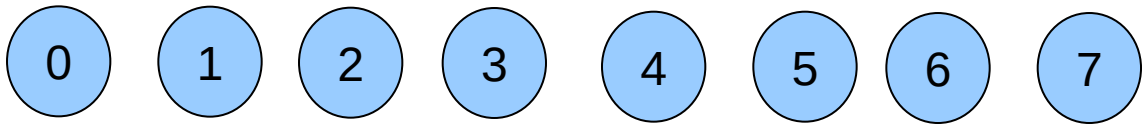


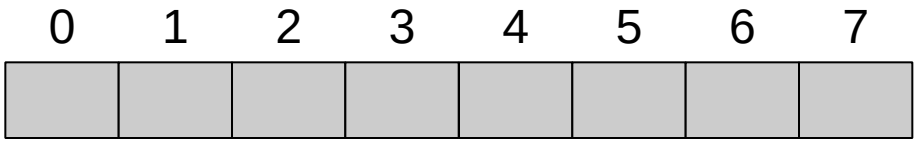
0	1	2	3	4	5	6	7	8	9
-1	-1	0	0	1	0	1	1	1	1
771	121	281	854	885	734	486	1003	83	62

III – Ensembles disjoints sous forme d'arbres

Exemple:

Prenons les nœuds suivants

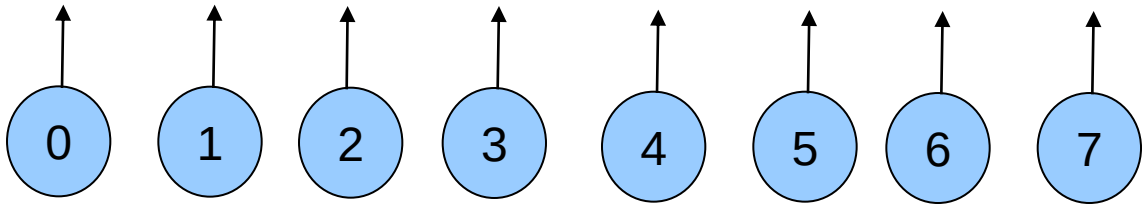




III – Ensembles disjoints sous forme d'arbres

Exemple:

On effectue la requête Create(a) sur l'ensemble des nœuds

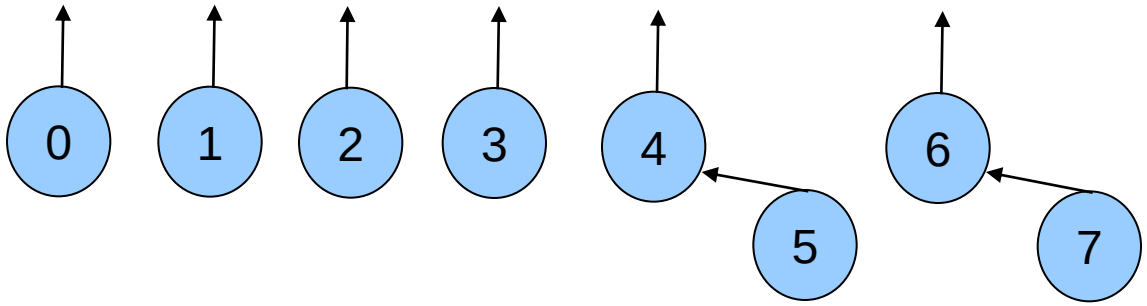


0	1	2	3	4	5	6	7
-1	-1	-1	-1	-1	-1	-1	-1

III – Ensembles disjoints sous forme d'arbres

Exemple:

On effectue Union(4, 5) puis Union(6, 7): Plus petite valeur devient racine (arbitraire).

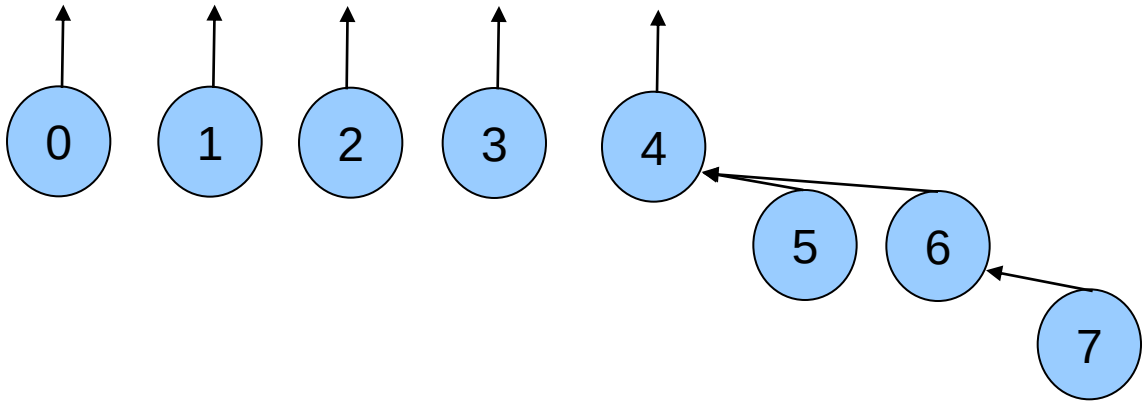


0	1	2	3	4	5	6	7
-1	-1	-1	-1	-1	4	-1	6

III – Ensembles disjoints sous forme d'arbres

Exemple:

On effectue Union(4, 6)

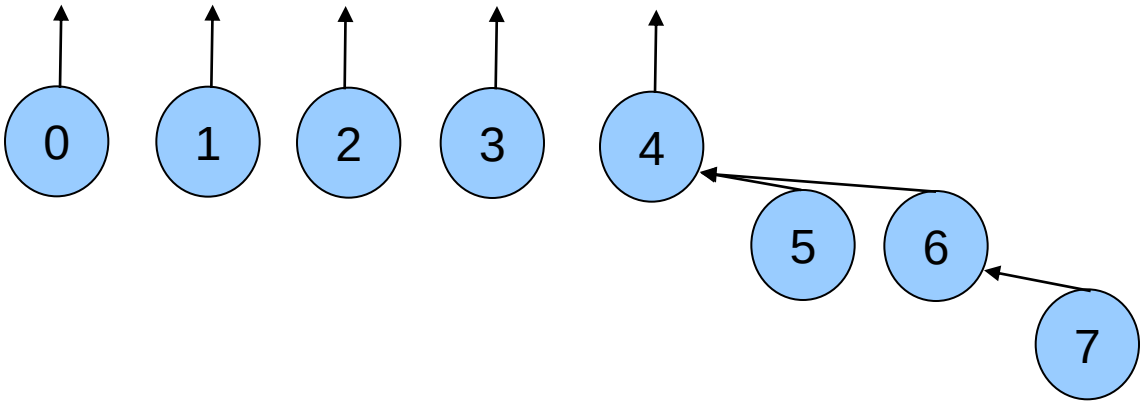


0	1	2	3	4	5	6	7
-1	-1	-1	-1	-1	4	4	6

III – Ensembles disjoints sous forme d'arbres

Exemple:

Il apparaît alors que l'opération Union(a, b) a une complexité O(1), tandis que Find(a) a une complexité O(n)

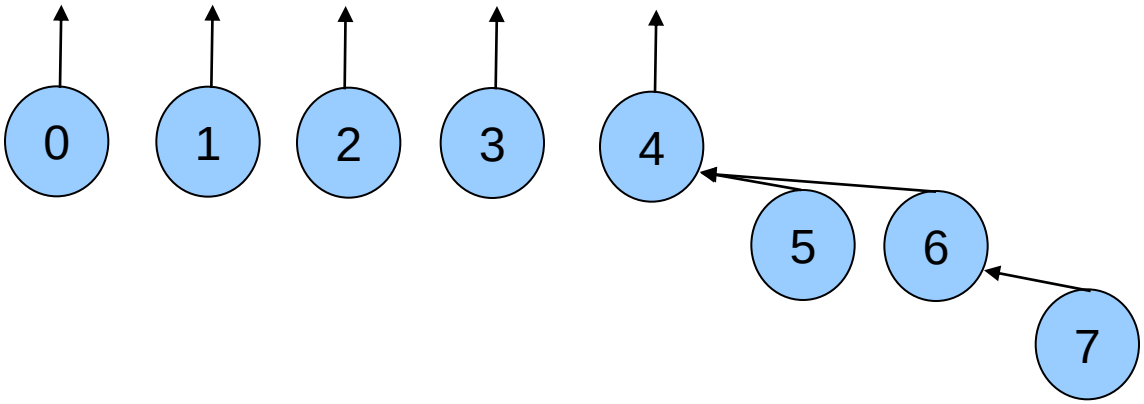


0	1	2	3	4	5	6	7
-1	-1	-1	-1	-1	4	4	6

III – Ensembles disjoints sous forme d'arbres

Améliorations:

Afin de réduire la complexité de la Find(a) de O(n) à O(lg(n)), on peut imposer des contraintes structurelles à l'opération d'union

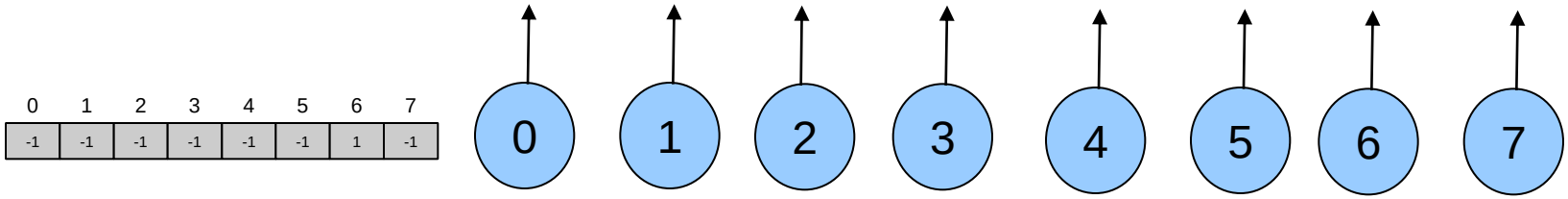


0	1	2	3	4	5	6	7
-1	-1	-1	-1	-1	4	4	6

III – Ensembles disjoints sous forme d'arbres

Amélioration 1:

Union par taille: lors de l'union de deux arbres, le plus petit des deux arbres devient l'enfant du plus grand:

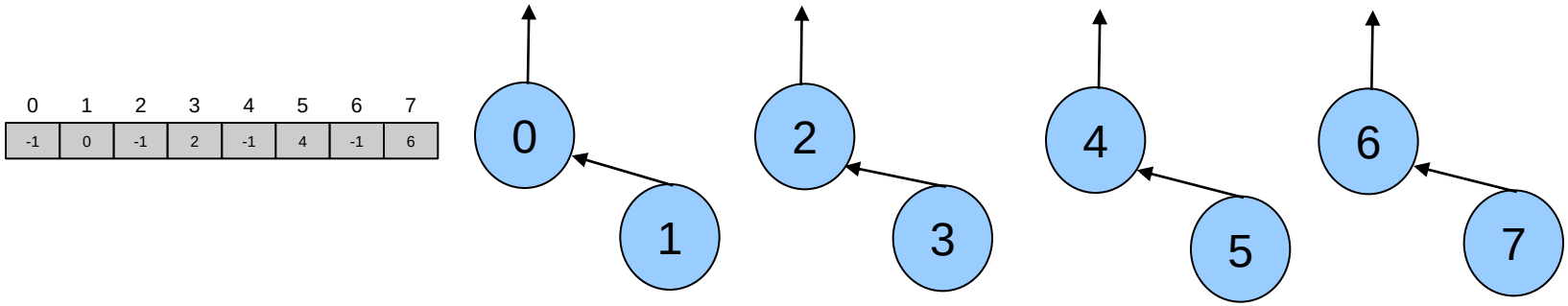


Taille	1	1	1	1	1	1	1	1

III – Ensembles disjoints sous forme d'arbres

Union par taille:

Union(0,1), Union(2,3) Union(4,5), Union(6,7)

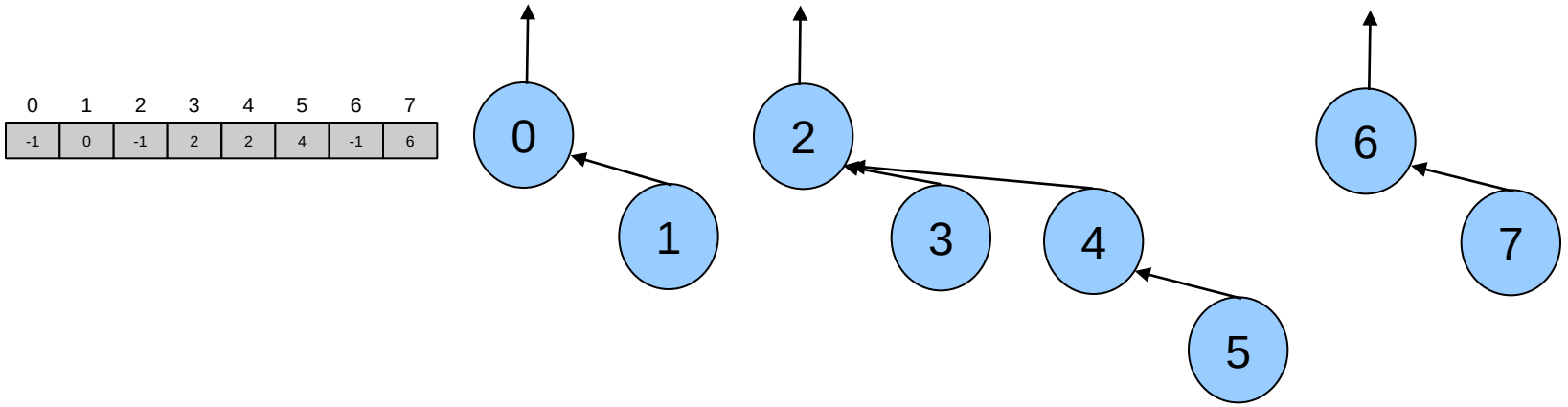


Taille	2	2	2	2

III – Ensembles disjoints sous forme d'arbres

Union par taille:

Union(2,4)



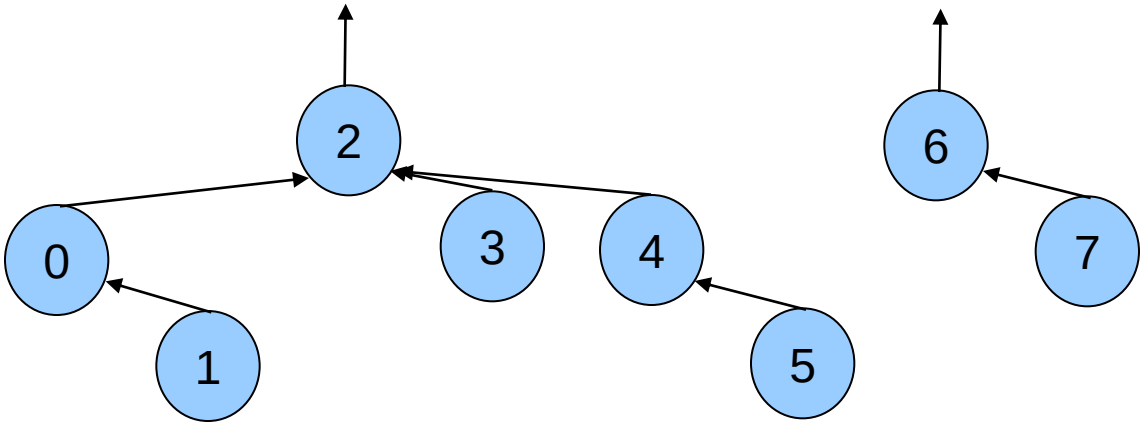
Taille 2 4 2

III – Ensembles disjoints sous forme d'arbres

Union par taille:

Union(0,2)

0	1	2	3	4	5	6	7
2	0	-1	2	2	4	-1	6



6 2

Plan

I. Formulation initiale
II. Définitions sur les ensembles disjoints
III. Ensembles disjoints sous forme d'arbres
IV. Compression de chemin
V. Algorithme de Kruskal

IV – Compression de chemin

Problématique:

Il est souvent préférable d'effectuer une analyse de complexité amortie plutôt que d'inspecter les opérations individuellement.

Afin d'amortir le coût de m requêtes de Find(a) et obtenir une complexité meilleure que O(m·lg(n)), on rectifie le chemin vers le parent.

IV – Compression de chemin

Code de compression du chemin:

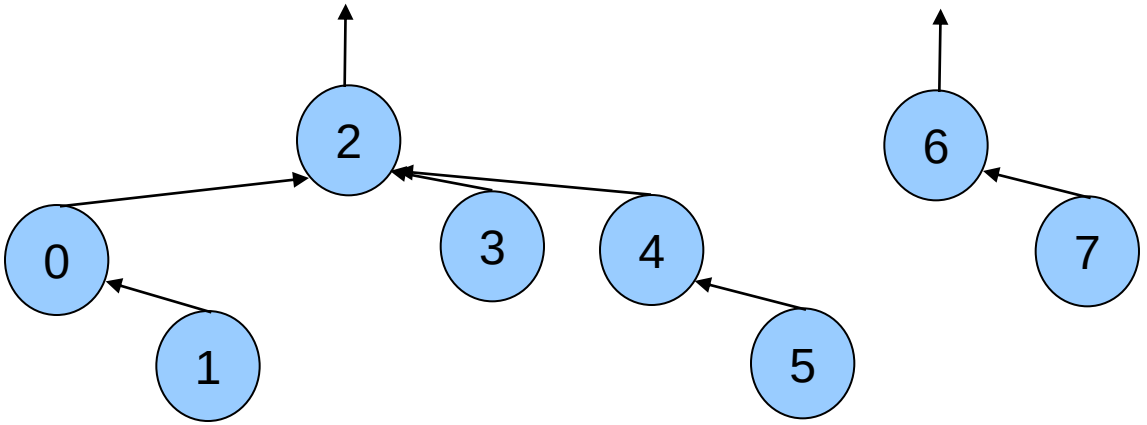
```
public int find( int x )
{

    if( s[ x ] < 0 )
        return x;
    else
        return s[x] = find( s[x] );

}
```

IV – Compression de chemin

Exemple

Partons de l'arbre précédent:

0	1	2	3	4	5	6	7
2	0	-1	2	2	4	-1	6

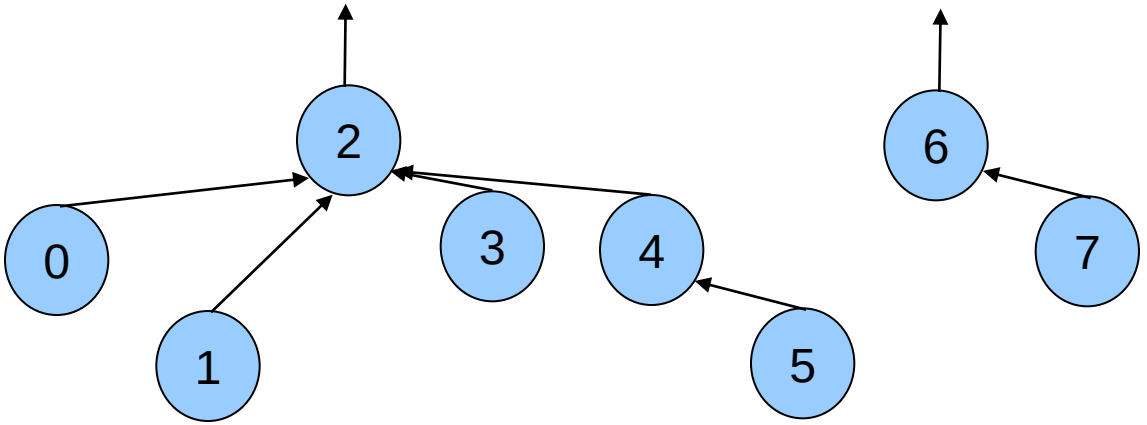


IV – Compression de chemin

Exemple

Find(1):

0	1	2	3	4	5	6	7
2	2	-1	2	2	4	-1	6

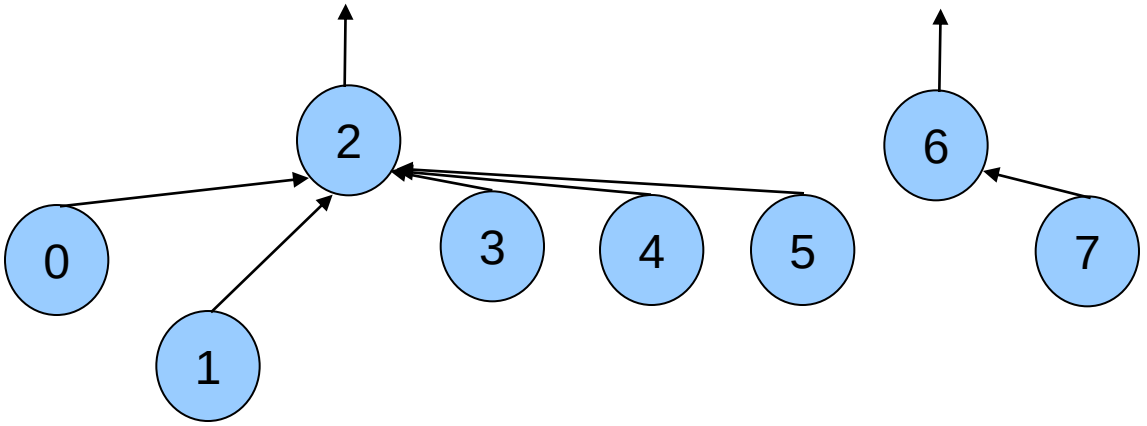


IV – Compression de chemin

Exemple

Find(5):

0	1	2	3	4	5	6	7
2	2	-1	2	2	2	-1	6



Plan

I. Formulation initiale
II. Définitions sur les ensembles disjoints
III. Ensembles disjoints sous forme d'arbres
IV. Compression de chemin
V. Algorithme de Kruskal

V – Algorithme de Kruskal

Définition

L'algorithme de Kruskal permet de trouver l'arbre sous-tendant minimum d'un graphe en utilisant la structure de données des ensemble disjoints.

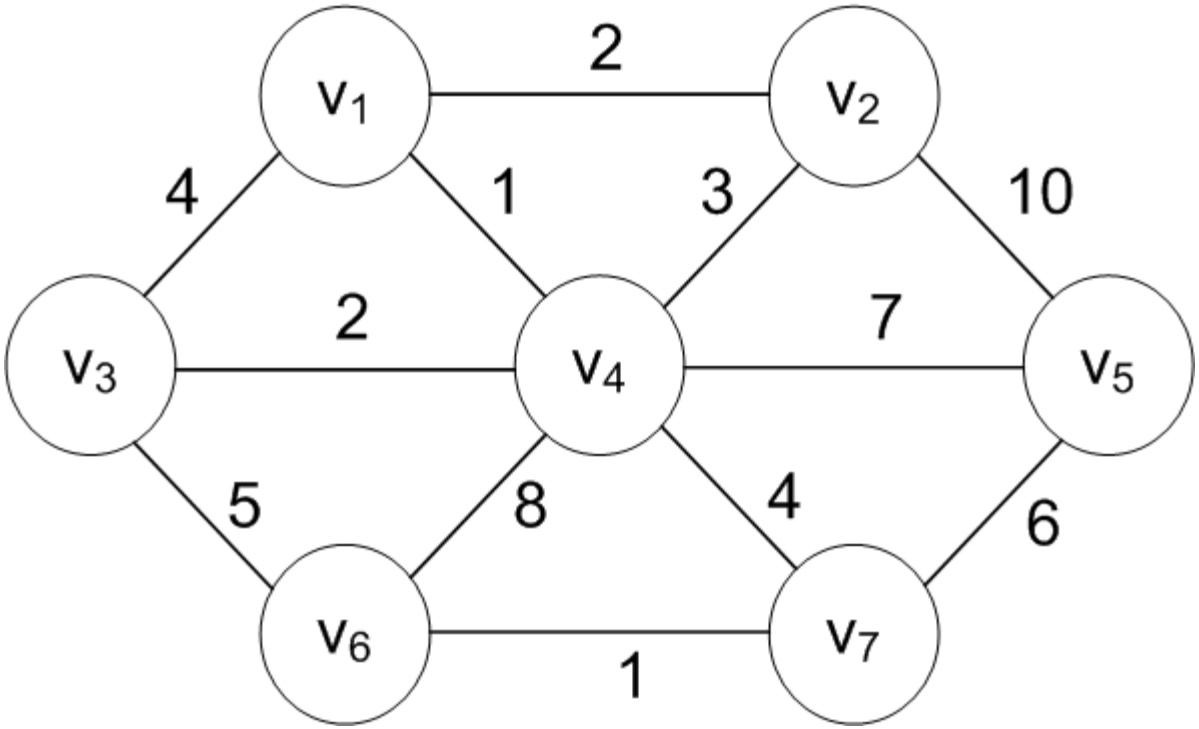
Il s'exprime comme suit:

Créer une forêt **F** à partir des sommets du graphe[1]
Créer une collection **A** contenant tous les arcs du graphe
Tant que **A** n'est pas vide et que **F** contient plus d'un arbre
    Retirer de **A** l'arc le plus petit poids
    Si l'arc lie deux arbres différents dans **F**,
        Union des deux arbres

[1] Chaque sommet constitue un arbre

V – Algorithme de Kruskal

Considérons le graphe valué et non dirigé suivant, pour lequel on cherche l'arbre sous-tendant minimum:



V – Algorithme de Kruskal

On cherche à obtenir ceci: