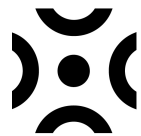
Algorithmique des arbres

Chercher-Réunir (Union-Find)

L2 Mathématique et Informatique



Objectif général

Le but est de gérer des ensembles disjoints.

Les différentes requêtes devant être implantées sont :

- `creer_ensemble(x)` : crée l'ensemble ne contenant que l'entier `x` ;
- `Trouver(x)` : à quel ensemble appartient x ?
- `Fusion(x,y)` : réunir les ensembles contenant x et y, si x et y ne sont pas déjà dans le même ensemble.

Pour chaque ensemble, on choisit un élément, le *représentant*, dont la valeur caractérise l'ensemble.

Simplification : Ensembles $\longrightarrow$ Ensembles d'entiers

Pour simplifier, on ne considérera ici que des ensembles d'entiers positifs inférieurs ou égal à un nombre `N`.

Pour le cas général, ces entiers peuvent servir d'indices dans un tableau de `N` éléments

Outline

1. Exemple d'application : construction d'un labyrinthe
2. Implémentation par listes chainées
3. Implémentation arborescente
   - Implantation "naïve"
   - Amélioration de la hauteur
   - Complexité

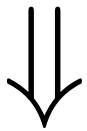
Construction d'un labyrinthe

0	1	2	3
4	5	6	7
8	9	10	11

Collection d'ensembles disjoints :

$\{0\}, \{1\}, \{2\}, \{3\}, \{4\}, \{5\},$
$\{6\}, \{7\}, \{8\}, \{9\}, \{10\}, \{11\}$

0 et 11 ne sont pas dans le même ensemble



le labyrinthe n'est pas entièrement construit.

Construction d'un labyrinthe

0	1	2	3
4	5	6	7
8	9	10	11

Collection d'ensembles disjoints :

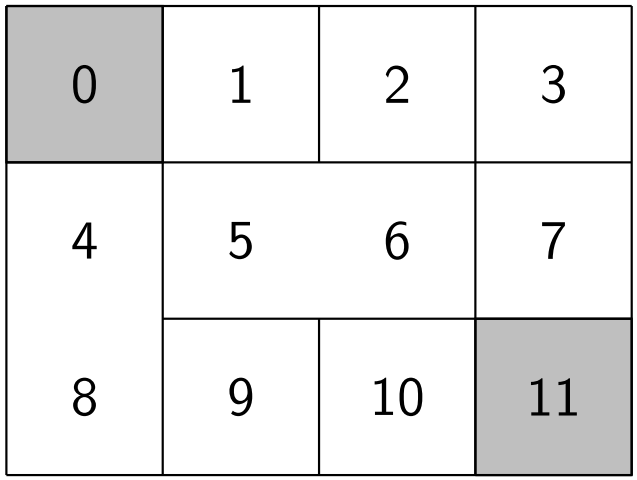
$\{0\}, \{1\}, \{2\}, \{3\}, \{4, 8\}, \{5\},$
$\{6\}, \{7\}, \{9\}, \{10\}, \{11\},$

Suppression du mur entre les cases 4 et 8.

0 et 11 ne sont pas dans le même ensemble

le labyrinthe n'est pas entièrement construit.

Construction d'un labyrinthe



Collection d'ensembles disjoints :

$\{0\}, \{1\}, \{2\}, \{3\}, \{4, 8\},$
$\{5, 6\}, \{7\}, \{9\}, \{10\}, \{11\},$

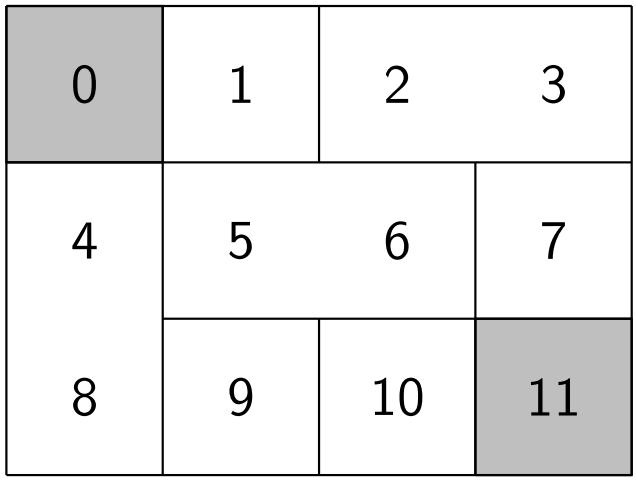
Suppression du mur entre les cases
5 et 6.

0 et 11 ne sont pas dans le même ensemble

$\Downarrow$

le labyrinthe n'est pas entièrement construit.

Construction d'un labyrinthe



Collection d'ensembles disjoints :

$\{0\}, \{1\}, \{2,3\}, \{4,8\},$
$\{5,6\}, \{7\}, \{9\}, \{10\}, \{11\},$

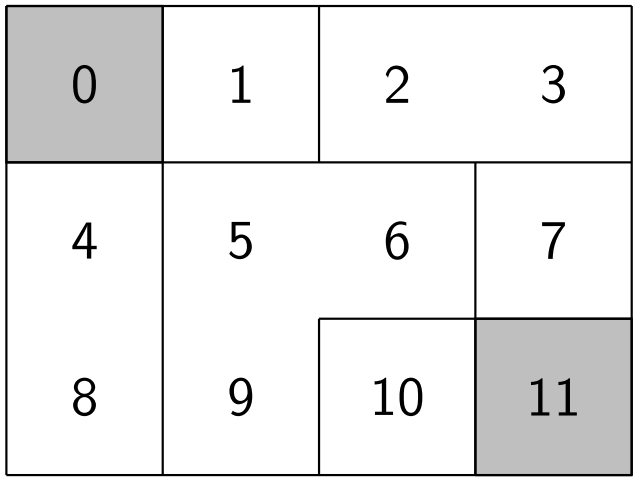
Suppression du mur entre les cases
2 et 3.

0 et 11 ne sont pas dans le même ensemble

$\Downarrow$

le labyrinthe n'est pas entièrement construit.

Construction d'un labyrinthe



Collection d'ensembles disjoints :

$\{0\}, \{1\}, \{2,3\}, \{4,8\},$
$\{5,6,9\}, \{7\}, \{10\}, \{11\},$

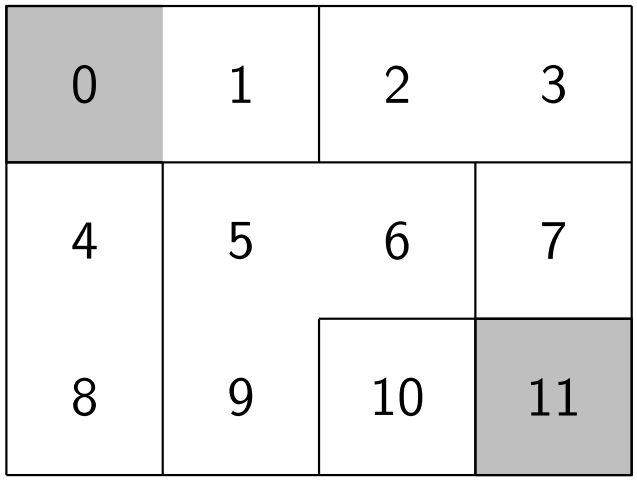
Suppression du mur entre les cases
5 et 9.

0 et 11 ne sont pas dans le même ensemble

$\Downarrow$

le labyrinthe n'est pas entièrement construit.

Construction d'un labyrinthe



Collection d'ensembles disjoints :

$\{0,1\}$, $\{2,3\}$, $\{4,8\}$,
$\{5,6,9\}$, $\{7\}$, $\{10\}$, $\{11\}$,

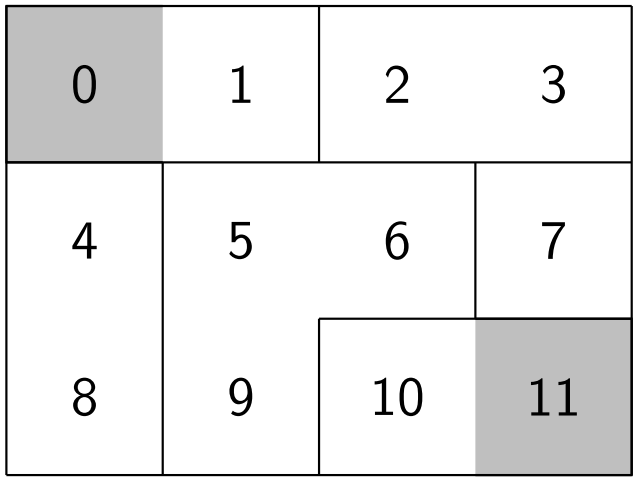
Suppression du mur entre les cases
0 et 1.

0 et 11 ne sont pas dans le même ensemble

$\Downarrow$

le labyrinthe n'est pas entièrement construit.

Construction d'un labyrinthe



Collection d'ensembles disjoints :

$\{0,1\}$, $\{2,3\}$, $\{4,8\}$,
$\{5,6,9\}$, $\{7\}$, $\{10,11\}$,

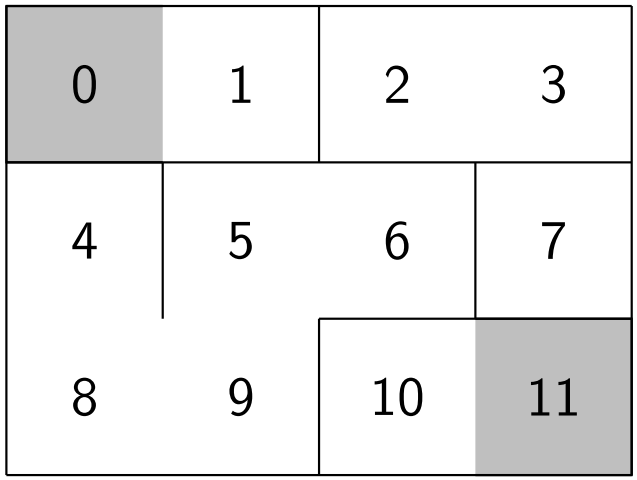
Suppression du mur entre les cases
10 et 11.

0 et 11 ne sont pas dans le même ensemble

$\Downarrow$

le labyrinthe n'est pas entièrement construit.

Construction d'un labyrinthe



Collection d'ensembles disjoints :

$\{0,1\}$, $\{2,3\}$, $\{4,5,6,8,9\}$,
$\{7\}$, $\{10,11\}$,

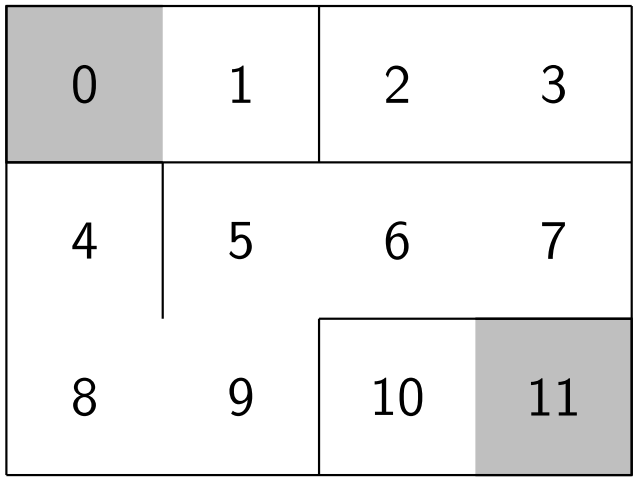
Suppression du mur entre les cases
8 et 9.

0 et 11 ne sont pas dans le même ensemble

$\Downarrow$

le labyrinthe n'est pas entièrement construit.

Construction d'un labyrinthe



Collection d'ensembles disjoints :

$\{0,1\}$, $\{2,3\}$, $\{4,5,6,7,8,9\}$, $\{10,11\}$,

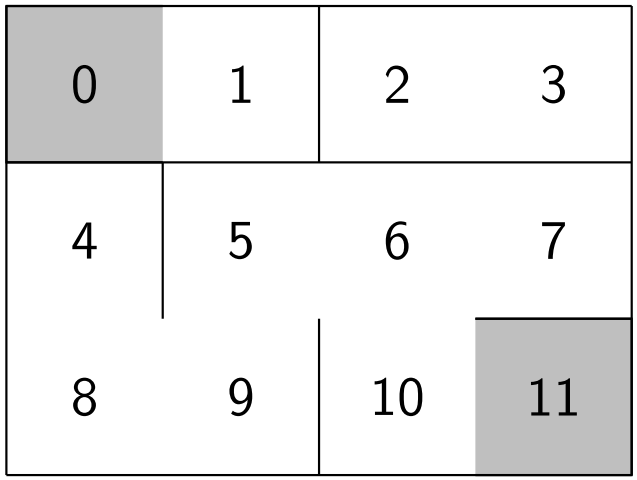
Suppression du mur entre les cases 6 et 7.

0 et 11 ne sont pas dans le même ensemble

$\Downarrow$

le labyrinthe n'est pas entièrement construit.

Construction d'un labyrinthe



Collection d'ensembles disjoints :

$\{0, 1\}$, $\{2, 3\}$,
$\{4, 5, 6, 7, 8, 9, 10, 11\}$

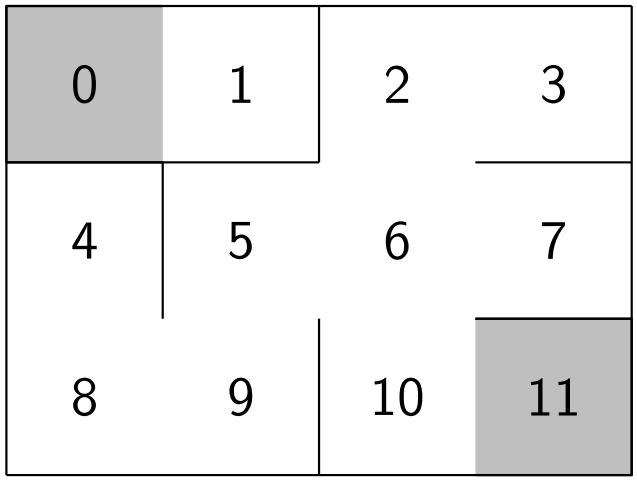
Suppression du mur entre les cases
6 et 10.

0 et 11 ne sont pas dans le même ensemble

$\Downarrow$

le labyrinthe n'est pas entièrement construit.

Construction d'un labyrinthe



Collection d'ensembles disjoints :

$\{0, 1\}$,
$\{2, 3, 4, 5, 6, 7, 8, 9, 10, 11\}$

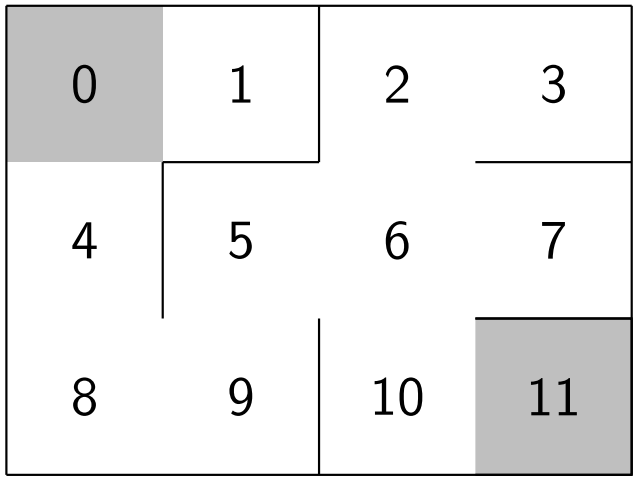
Suppression du mur entre les cases
2 et 6.

0 et 11 ne sont pas dans le même ensemble

$\Downarrow$

le labyrinthe n'est pas entièrement construit.

Construction d'un labyrinthe



Collection d'ensembles disjoints :

$\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11\}$

Suppression du mur entre les cases
0 et 4.

0 **et** 11 **sont désormais dans le même ensemble**

$\Downarrow$

le labyrinthe npeut être considéré comme entièrement construit.

Outline

1 Exemple d'application : construction d'un labyrinthe

2 Implémentation par listes chainées

3 Implémentation arborescente
- Implantation "naïve"
- Amélioration de la hauteur
- Complexité

Méthode

On organise les éléments des sous-ensembles sous forme de liste chainée :

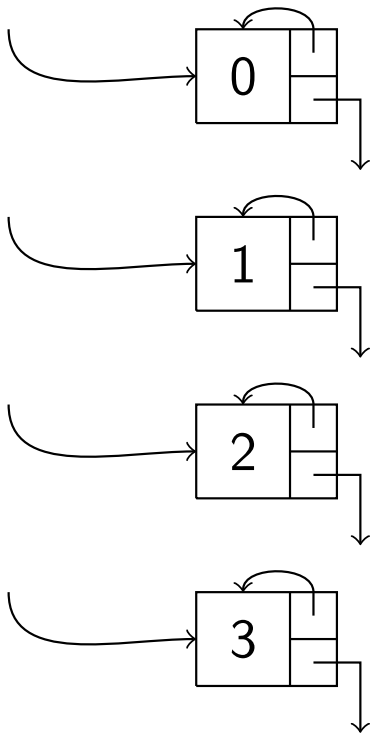
- chaque cellule contient un élément de l'ensemble ;
- la première cellule est le représentant de l'ensemble.

Comment retrouver le représentant ?

Chaque maillon possède un lien sur la tête de la liste.

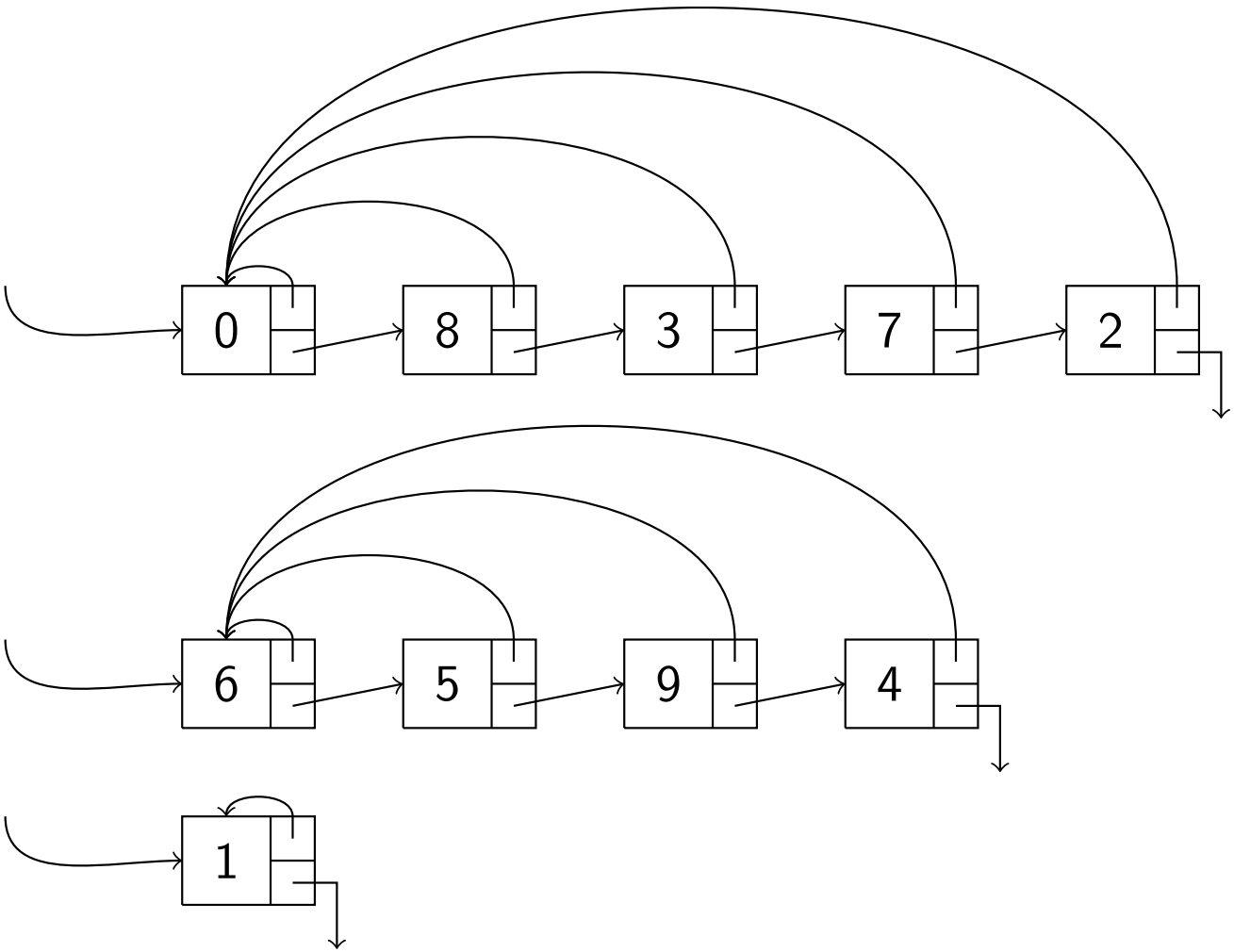
Exemple 1

Les ensembles $\{0\}, \{1\}, \{2\}, \{3\}$ correspondent aux listes chainées :



Exemple 2

Les ensembles $\{0, 2, 3, 7, 8\}, \{1\}, \{4, 5, 6, 9\}$) correspondent, par exemple, aux listes chainées :



Implantation

```
typedef struct cel {
    int valeur;
    struct cel * suivant:
    struct cel * representant;
} Cellule, * Liste;

Cellule * creer_ensemble(int x);
Cellule * Trouver(Cellule * elt);
void Fusion(Cellule * x, Cellule * y);

Cellule * Ensembles[N];
```

Trouver

Objectif : Trouver le numéro de l'ensemble auquel appartient un élément

Il suffit de suivre le lien vers la tête de la liste chainée :

```
Cellule * Trouver(Cellule * elt){
    return elt->representant;
}
```

La complexité est en $\mathcal{O}(1)$.

Fusion

Objectif : Fusionner les ensembles contenant les éléments x et y.

- Déterminer r_x, le représentant de l'ensemble contenant x ;
- Déterminer r_y,le représentant de l'ensemble contenant y ;
- Si x et y sont dans le même ensemble (`r_x == r_y`) :

  il n'y a rien à faire ;

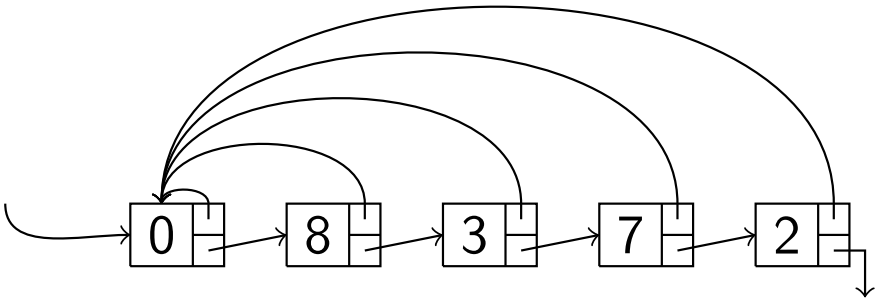
  Sinon, concaténer les deux listes chainées et mettre à jour les représentants.

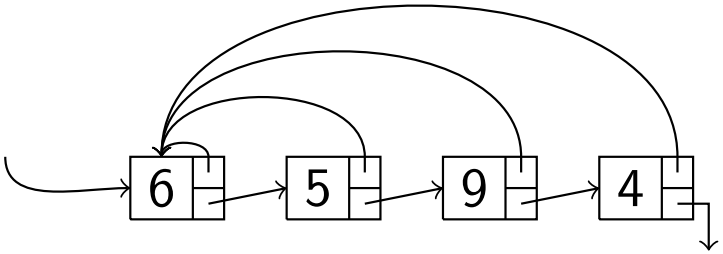
Exercice de révision : Ecrire la fonction `fusion`

La fusion s'effectue donc en $\mathcal{O}(\mathtt{N})$ dans le pire des cas.

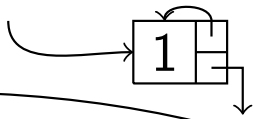
Exemple de fusion

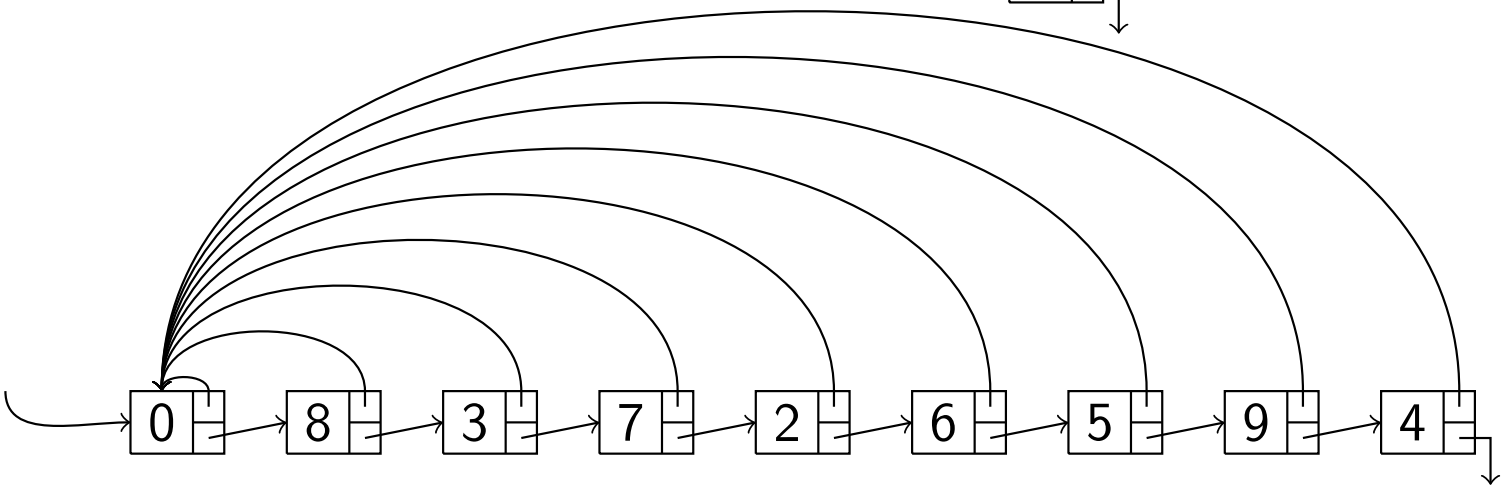
On reprends l'exemple 2 précédent.

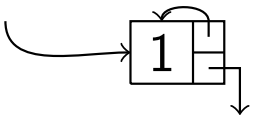




On souhaite fusionner l'ensemble contenant 2 avec celui contenant 6.







Compléxité avec l'union par rang et la compression des chemins

Propriété :

Soit E une structure `Chercher-Trouver` de n entiers.
Il existe une suite de `m` opérations (`creer_ensemble`, `Trouver` ou `Fusion`) sur les `n` éléments gérés par E telle que la complexité de celles-ci soit $\mathcal{O}(\mathtt{N}^2)$.

Exemple :

```
Cellule * Ensemble[N];
Ensemble[0] = creer_ensemble(1);
Ensemble[1] = creer_ensemble(2);
⋮
Ensemble[N - 1] = creer_ensemble(N);
Fusion(Ensemble[0], Ensemble[1]);
Fusion(Ensemble[0], Ensemble[2]);
⋮
Fusion(Ensemble[0], Ensemble[N - 1]);
```

Outline

1 Exemple d'application : construction d'un labyrinthe

2 Implémentation par listes chainées

3 Implémentation arborescente
- Implantation "naïve"
- Amélioration de la hauteur
- Complexité

Outline

1 Exemple d'application : construction d'un labyrinthe

2 Implémentation par listes chainées

3 Implémentation arborescente
- Implantation "naïve"
- Amélioration de la hauteur
- Complexité

Méthode

On organise les éléments des sous-ensembles en arbre :

- chaque nœud contient un élément de l'ensemble ;
- l'étiquette de la racine est le représentant de l'ensemble.

Comment retrouver le représentant ?

Besoin de remonter des nœuds vers la racine

$\Longrightarrow$ Implantation des arbres par indice dans un tableau.

Convention : parent(`racine`) = `racine elle-même`.

Exemple 1

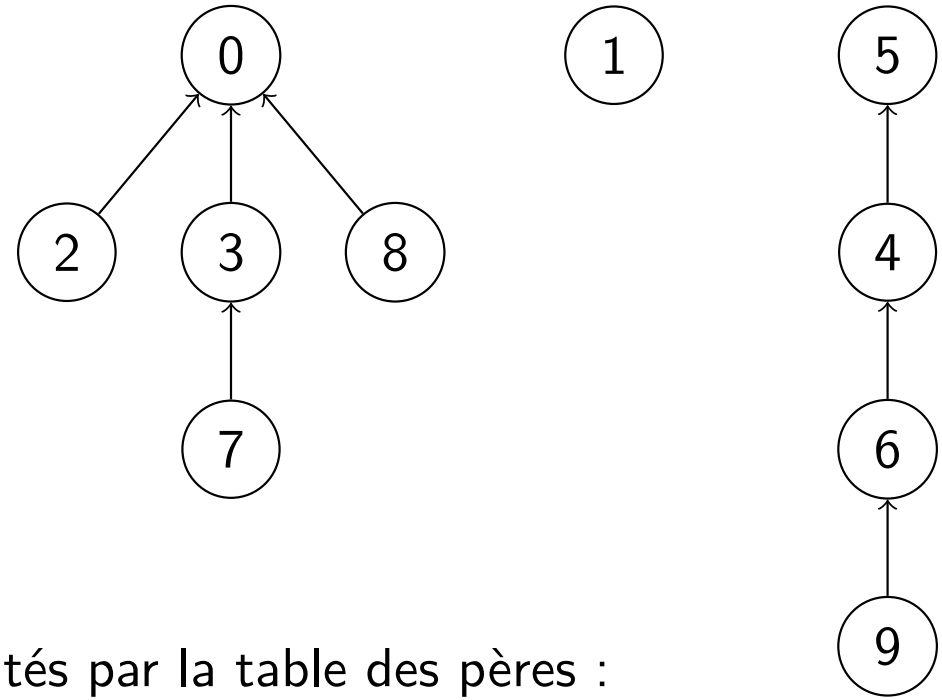
Les ensembles $\{0\}, \{1\}, \{2\}, \{3\}, \{4\}, \{5\}, \{6\}$ correspondent aux arbres :

Ils sont représentés par la table des pères :

Indice	0	1	2	3	4	5	6
Parent	0	1	2	3	4	5	6

Exemple 2

Les ensembles $\{0, 2, 3, 7, 8\}, \{1\}, \{4, 5, 6, 9\}$) correspondent, par exemple, aux arbres :



Ils sont représentés par la table des pères :

Indice	0	1	2	3	4	5	6	7	8	9
Parent	0	1	0	0	5	5	4	3	0	6

Le nœud 6 a le même représentant que son père 4.
Le nœud 4 a le même représentant que son père 5.
Le nœud 5 est la racine et a pour représentant 5.
$\Longrightarrow$ 6 est représenté par 5.

Trouver

Objectif : Trouver le numéro de l'ensemble auquel appartient un élément

Il suffit de remonter jusqu'à la racine :

```
int Trouver(int peres[], int element){
    while(peres[element] != element) {
        element = peres[element];
    }
    return element;
}
```

La complexité est en $\mathcal{O}$(hauteur de l'arbre).

Fusion

Objectif : Fusionner les ensembles contenant les éléments x et y.

- Déterminer n_x, numéro de l'ensemble contenant x ;
- Déterminer n_y, numéro de l'ensemble contenant y ;
- Si x et y sont dans le même ensemble (`n_x == n_y`) :

  il n'y a rien à faire ;

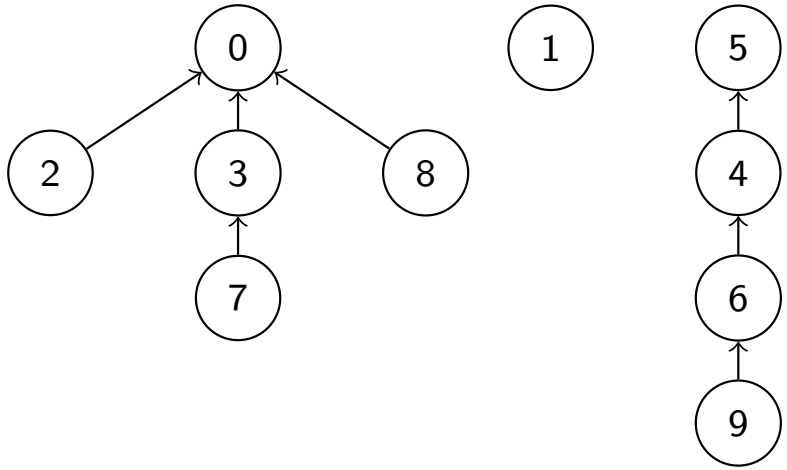
  Sinon, la racine d'un des arbres devient le père de la racine de l'autre.

```
void Fusion(int peres[], int x, int y){
    int n_x = Trouver(peres, x);
    int n_y = Trouver(peres, y);
    if (n_x != n_y)
        peres[n_y] = n_x;
}
```

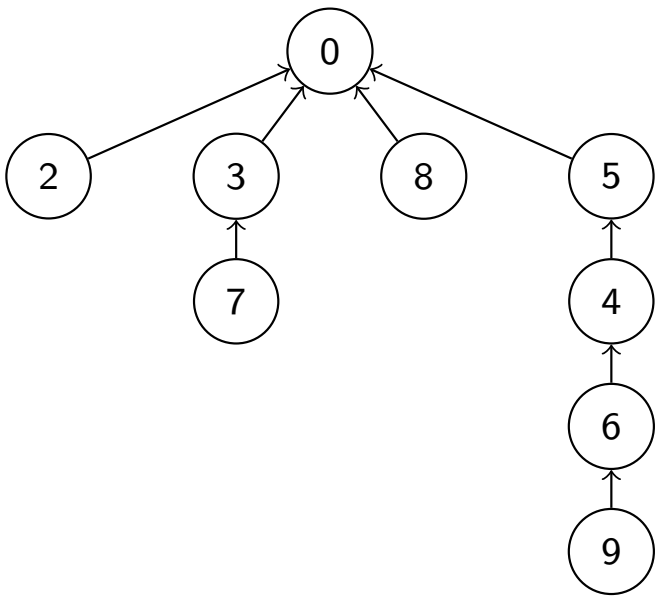
La fusion s'effectue donc en $\mathcal{O}$(hauteur de l'arbre).

Exemple de fusion

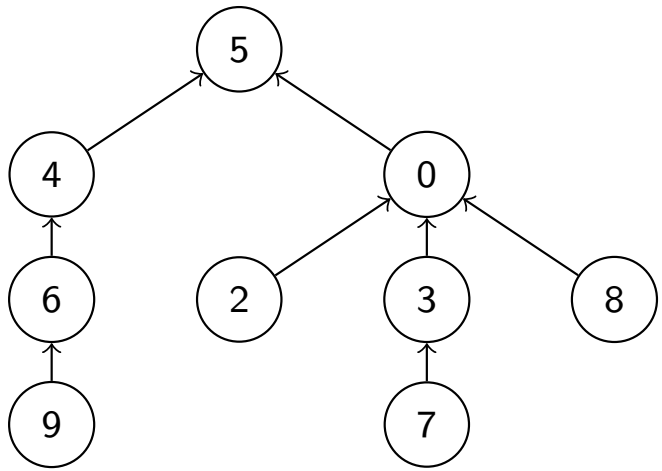
On reprends l'exemple 2 précédent :



On souhaite fusionner l'ensemble contenant 2 avec celui contenant 6.



ou



Exemple de fusion

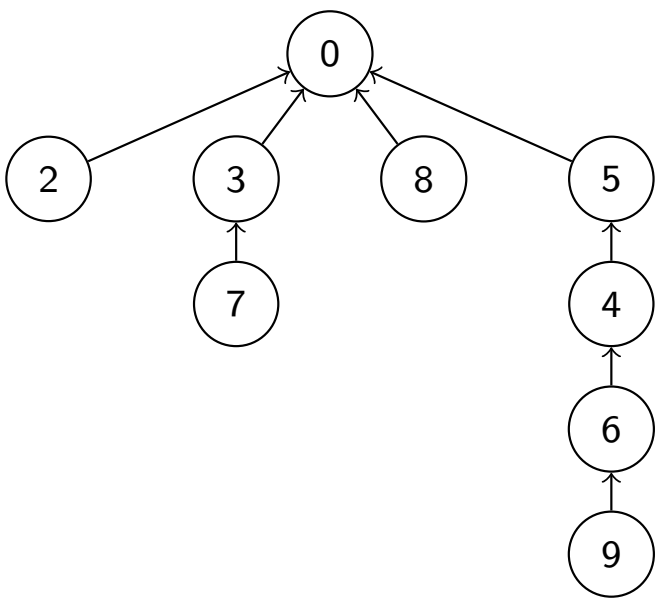
Tables des pères :

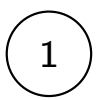
Indice	0	1	2	3	4	5	6	7	8	9
Parent	0	1	0	0	5	0	4	3	0	6

ou

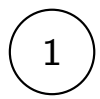
Indice	0	1	2	3	4	5	6	7	8	9
Parent	5	1	0	0	5	5	4	3	0	6

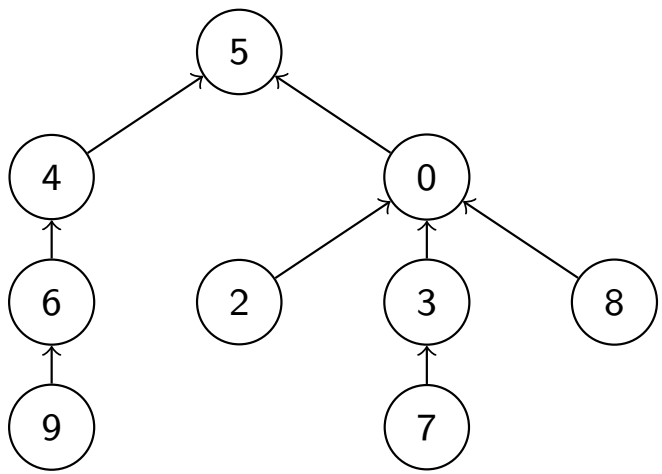
Hauteur max des arbres : 4 contre 3





ou





Outline

1 Exemple d'application : construction d'un labyrinthe

2 Implémentation par listes chainées

3 Implémentation arborescente
- Implantation "naïve"
- Amélioration de la hauteur
- Complexité

Diminution de la hauteur lors de la fusion

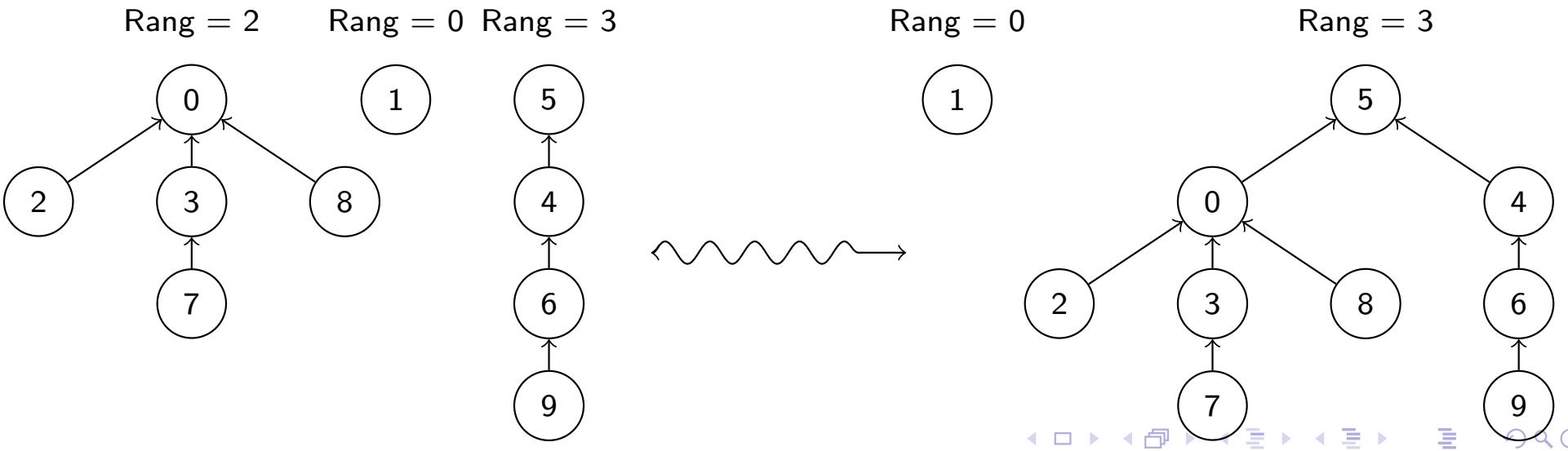
Sans précaution, la hauteur de l'arbre peut atteindre $n - 1$.

Comment diminuer la hauteur de l'arbre ?

Deux méthodes sont utilisées simultanément :

1. **l'union par rang :** Rang d'un arbre = borne sup. de sa hauteur.

   On conserve le *rang* de chaque arbre : lors d'une fusion, la racine de l'arbre de plus haut rang devient celle du nouvel arbre.
   $\Longrightarrow$ La hauteur maximale des arbres n'augmente au plus que de 1.



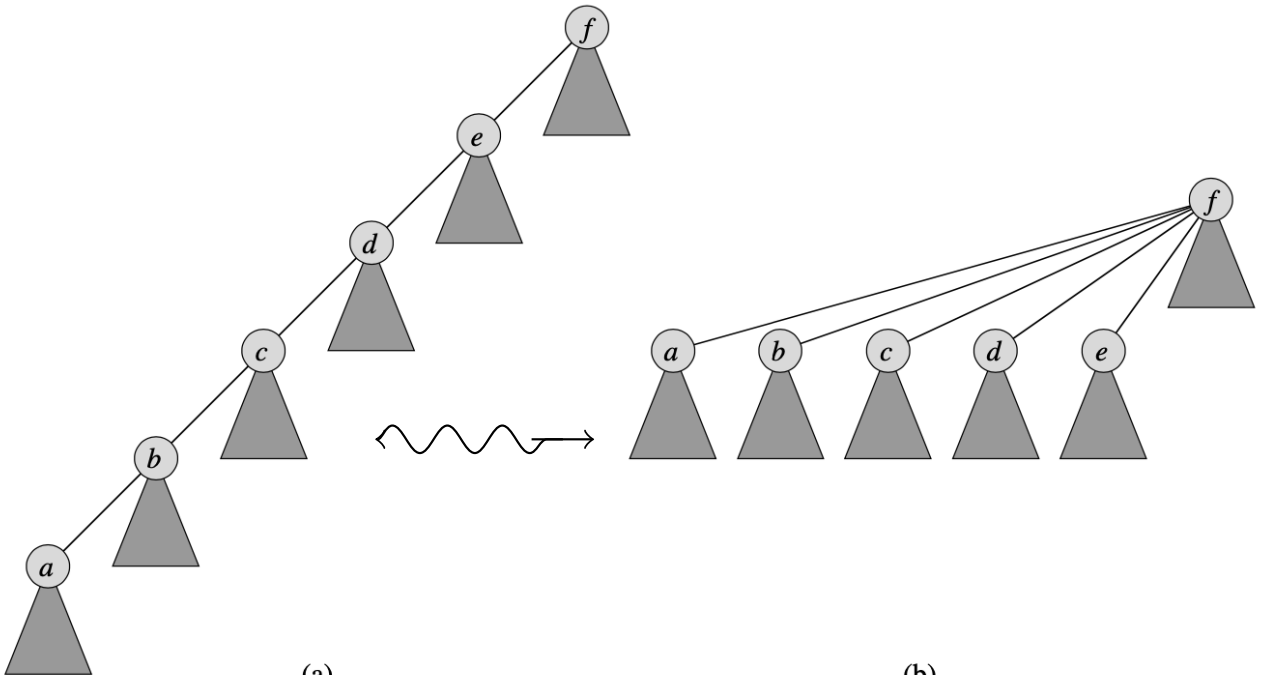
Diminution de la hauteur lors de la fusion

Sans précaution, la hauteur de l'arbre peut atteindre $n - 1$.

Comment diminuer la hauteur de l'arbre ?

Deux méthodes sont utilisées simultanément :

2. **la compression des chemins :**
   Lorsqu'on recherche le numéro d'un élément, on fait de la racine le père de tous les nœuds rencontrés.
   La compression de chemin ne modifie pas le rang.



Diminution de la hauteur lors de la fusion

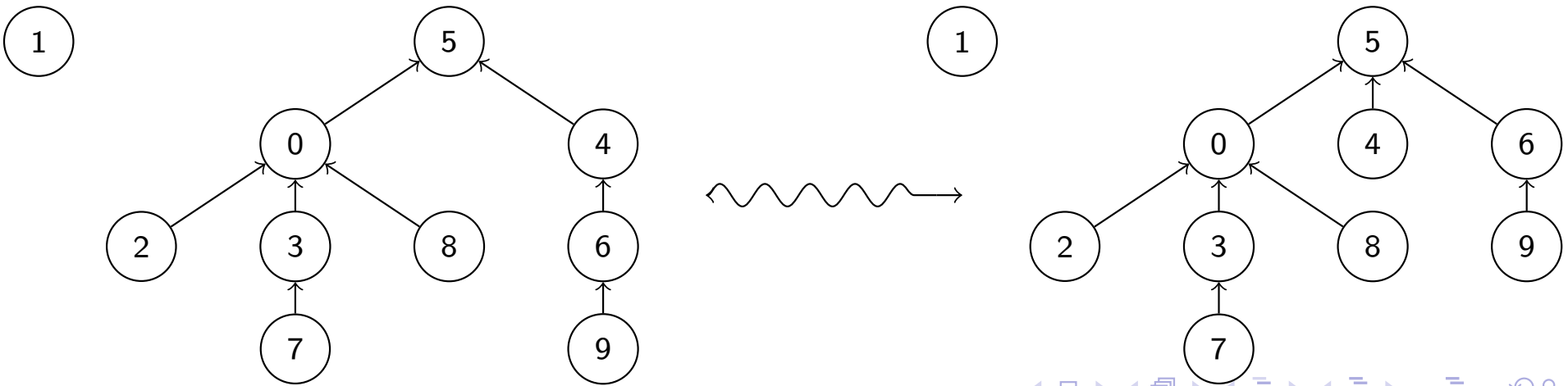
Sans précaution, la hauteur de l'arbre peut atteindre $n - 1$.

Comment diminuer la hauteur de l'arbre ?

Deux méthodes sont utilisées simultanément :

2. **la compression des chemins :**
   Lorsqu'on recherche le numéro d'un élément, on fait de la racine le père de tous les nœuds rencontrés.
   La compression de chemin ne modifie pas le rang.

Recherche du numéro de l'ensemble contenant 6 :



Nouvelle implémentation de Trouver

Ancienne version :

```
int Trouver(int peres[], int element){
    while(peres[element] != element) {
        element = peres[element];
    }
    return element;
}
```

Nouvelle version (avec compression des chemins) :

```
int Trouver(int peres[], int element){
    if (peres[element] != element)
        peres[element] = Trouver(peres,peres[element]);
    return peres[element];
}
```

Diminution de la hauteur lors de la fusion : Résumé

Sans précaution, la hauteur de l'arbre peut atteindre $n - 1$.

Comment diminuer la hauteur de l'arbre ?

Deux méthodes sont utilisées simultanément :

1. **l'union par rang :** Rang d'un arbre = borne sup. de sa hauteur.

   On conserve le *rang* de chaque arbre : lors d'une fusion, la racine de l'arbre de plus haut rang devient celle du nouvel arbre.

   $\Longrightarrow$ La hauteur maximale des arbres n'augmente au plus que de 1.

2. **la compression des chemins :**

   Lorsqu'on recherche le numéro d'un élément, on fait de la racine le père de tous les nœuds rencontrés.

   La compression de chemin ne modifie pas le rang.

Outline

1 Exemple d'application : construction d'un labyrinthe

2 Implémentation par listes chainées

3 Implémentation arborescente
- Implantation "naïve"
- Amélioration de la hauteur
- Complexité

Compléxité avec l'union par rang et la compression des chemins

Propriété :

Soit E une structure `Chercher-Trouver` de n entiers.
Si l'on fait `m` opérations (`creer_ensemble`, `Trouver` ou `Fusion`) sur les `n` éléments gérés par E, alors la complexité de celles-ci est en $\mathcal{O}\big(m \cdot \alpha(m, n)\big)$.

Remarque : Il faut au moins créer les n ensembles à 1 élément, donc, nécessairement, $m \geq n$.

La fonction $\alpha(m, n)$ est *presque* l'inverse de la fonction d'Ackerman.

Remarque : $\alpha(n, m)$ est une fonction à croissance tellement lente qu'on peut la considérer quasi constante.

Fonction d'Ackerman

Définition :

La fonction d'Ackerman est définie pour des entiers $i, j \geq 1$ par :

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

Propriété simple 1 : La fonction d'Ackermann est correctement définie sur $\mathbb{N}^\star \times \mathbb{N}^\star$.

Propriété simple 2 : Toute évaluation de la fonction d'Ackermann est une puissance de 2.

Preuves par induction bien fondée sur l'ordre lexicographique.

Fonction d'Ackerman

Définition :

La fonction d'Ackerman est définie pour des entiers $i, j \geq 1$ par :

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

Exemples :

$$A(1,n) = 2^n$$

Fonction d'Ackerman

Définition :

La fonction d'Ackerman est définie pour des entiers $i, j \geq 1$ par :

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

Exemples :

$$\begin{aligned} A(2,1) &= A(1,2) = 2^2 \\ A(2,2) &= A\big(1, A(2,1)\big) = A(1,2^2) = 2^{2^2} = 2^4 = 16 \\ A(2,3) &= A\big(1, A(2,2)\big) = A\big(1,2^{2^2}\big) = 2^{2^{2^2}} = 2^{16} = 65536 \\ A(2,n) &= 2^{2^{\cdot^{\cdot^{\cdot^2}}}}, \textit{tour de } n \textit{ exposants} \end{aligned}$$

Fonction d'Ackerman

Définition :

La fonction d'Ackerman est définie pour des entiers $i, j \geq 1$ par :

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

Exemples : $A(2,n) = 2^{2^{\cdot^{\cdot^{\cdot^{2}}}}}$ *, tour de n exposants*

$$A(3,1) = A(2,2) = 2^{2^2} = 65536$$

$$A(3,2) = A\big(2, A(3,1)\big) = A(2,16) = 2^{2^{\cdot^{\cdot^{\cdot^{2}}}}} \text{ , tour de 16 exposants}$$

$$A(3,3) = A\big(2, A(3,2)\big) = A\left(2, 2^{2^{\cdot^{\cdot^{\cdot^{2}}}}} \text{ , tour de 16 exposants}\right)$$

$$= 2^{2^{\cdot^{\cdot^{\cdot^{2}}}}} \text{ , tour de } 2^{2^{\cdot^{\cdot^{\cdot^{2}}}}} \text{ , tour de 16 exposants exposants}$$

Fonction d'Ackerman

Définition :

La fonction d'Ackerman est définie pour des entiers $i, j \geq 1$ par :

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

La fonction d'Ackermann crit extrêmement vite !!!

Notations de Knuth

Notations de Knuth :

$$a \uparrow^n b = \begin{cases} a^b & , \text{ si } n = 1 \\ 1 & , \text{ si } b = 0 \\ a \uparrow^{n-1} \left(a \uparrow^n (b-1)\right) & , \text{ sinon.} \end{cases}$$

Remarque : Voir le lien avec $A(i,j) = A(i-1, A(i,j-1))$

Exemples :

$$\begin{aligned} a \uparrow b &= a^b \\ a \uparrow^2 b &= a \uparrow \left(a \uparrow^2 (b-1)\right) = a \uparrow \left(a \uparrow \left(a \uparrow^2 (b-2)\right)\right) \\ &= \cdots = \underbrace{a \uparrow \left(a \uparrow \left(a \uparrow (\cdots \uparrow a)\right) \cdots\right)}_{b \text{ symbols } a} \\ a \uparrow^3 b &= a \uparrow^2 \left(a \uparrow^2 (b-1)\right) = a \uparrow^2 \left(a \uparrow^2 \left(a \uparrow^2 (b-1)\right)\right) \\ &= \cdots = \underbrace{a \uparrow\uparrow \left(a \uparrow\uparrow \left(a \uparrow\uparrow (\cdots \uparrow\uparrow a)\right) \cdots\right)}_{b \text{ symbols } a} \end{aligned}$$

Notations de Knuth

Notations de Knuth :

$$a \uparrow^n b = \begin{cases} a^b & , \text{ si } n = 1 \\ 1 & , \text{ si } b = 0 \\ a \uparrow^{n-1} \left(a \uparrow^n (b-1)\right) & , \text{ sinon.} \end{cases}$$

Remarque : Voir le lien avec $A(i,j) = A(i-1, A(i,j-1))$

Idée générale derrière la notation :

Knuth a inventée la notation $\uparrow$ pour représenter des nombres extrêmement grand,s sur le principe suivant :

une addition est une répétition d'additions de 1 ;
une multiplication est la répétition d'une addition ;
l'exponentiation est la répétition d'une multiplication.

Pourquoi ne pas continuer le processus ?

Notations de Knuth

Notations de Knuth :

$$a \uparrow^n b = \begin{cases} a^b & , \text{ si } n = 1 \\ 1 & , \text{ si } b = 0 \\ a \uparrow^{n-1} \left(a \uparrow^n (b-1)\right) & , \text{ sinon.} \end{cases}$$

Remarque : Voir le lien avec $A(i,j) = A(i-1, A(i,j-1))$

Fait 1 : $A(1,n) = 2 \uparrow n$ pour tout $n \in \mathbb{N}^\star$.

Fait 2 : $A(2,n) = 2 \uparrow^2 (n+1)$ pour tout $n \in \mathbb{N}^\star$.

Notations de Knuth

Notations de Knuth :

$$a \uparrow^n b = \begin{cases} a^b & , \text{ si } n = 1 \\ 1 & , \text{ si } b = 0 \\ a \uparrow^{n-1} \left(a \uparrow^n (b-1)\right) & , \text{ sinon.} \end{cases}$$

Remarque : Voir le lien avec $A(i,j) = A(i-1, A(i,j-1))$

Fait 1 : $A(1,n) = 2 \uparrow n$ pour tout $n \in \mathbb{N}^\star$.

Fait 2 : $A(2,n) = 2 \uparrow^2 (n+1)$ pour tout $n \in \mathbb{N}^\star$.

Une version modifiée $\widetilde{A}$ de la fonction d'Ackermann vérifie :

$$\widetilde{A}(i,j) = 2 \uparrow^{i-2} (j+3) - 3$$

Inverse de la fonction d'Ackermann : la fonction α

Definition :

Soit E la fonction partie entière. La fonction α est définie par :

$$\alpha(m,n) = \min\left\{ i \geq 1 \ , \ A\left(i, E(\frac{m}{n})\right) > \log(n) \right\}$$

Rappel : $m \geq n$.

$$\begin{aligned} A\left(4, E(\frac{m}{n})\right) &\geq A(4,1) = 2^{2^{\cdot^{\cdot^{\cdot^{2}}}}} \text{, tour de 16 exposants} \\ &\gg 2^{2^{2^{2^2}}} = 2^{2^{16}} = 2^{65536} = 2^6 \cdot (2^{10})^{6553} \\ &\approx 64 \cdot 10^{3 \cdot 6553} \gg 10^{80} \end{aligned}$$

Rappel : $2^{10} \approx 10^3$.

Rappel : $10^{80} \approx$ nombre d'atomes présents dans l'univers observable.

Donc, $A(4,1) \leq \log(n)$ pour des valeurs de n telles qu'on ne les rencontre jamais en pratique.