Analyse Amortie

Andrey Martinez Cruz

Table des matières

Analyse classique

À date ce qu'on fait c'est de l'analyse meilleur, moyen et pire des cas.
Conclusion complexité : pire des cas.

Analyse exagérée ?

Des fois, c'est souvent exagéré comme complexité, car arrive très rarement.
Exemple :

1. quicksort : $\mathcal{O}(n^2)$
2. inserer et regarder un élément dans une table de hachage (même avec une bonne fonction de hachage) : $\mathcal{O}(n)$

Analyse amortie

Définition : une analyse amortie consiste à évaluer la complexité sur une séquence de n opérations que plutôt sur un cas individuel.

Pire cas et meilleur cas : cas particulier.

Cas moyen $\neq$ Analyse amortie.

Méthodes

Les méthodes utilisés pour l'analyse amortie :

- Analyse de l'agrégat

Méthodes

Les méthodes utilisés pour l'analyse amortie :

- Analyse de l'agrégat
- Méthode comptable

Méthodes

Les méthodes utilisés pour l'analyse amortie :

- Analyse de l'agrégat
- Méthode comptable
- Méthode du potentiel

Analyse de l'agrégat

Soit $T(n) =$ sommation des coûts d'une opération sur une séquence de n opérations où les opérations sont les pires cas. Le coût amortie est représenté par :

$$ca = \frac{T(n)}{n}$$

Méthode comptable

Pour chaque opération, on peut lui attribuer un certain coût noté $ca(i)$, qui servira de crédit de réserve. Si $ca(i) > cr(i)$, on soustrait les crédits accumulés avec le coût réel de l'opération ($cr(i)$). Chaque opération peuvent avoir leur propres coûts amorties. En d'autres termes, le coût total qui est gardé dans la structure de donnée est

$$\sum_{i=0}^{n} ca(i) - \sum_{i=0}^{n} cr(i) \geq 0$$

Méthode du potentiel

On attribue le contenu de la structure de donnée ou du crédit (méthode du comptable) comme un "potentiel".

$\Phi(D_i)$: une fonction de potentiel permettant de dire l' "état" de la structure de donnée après la $i$nième opération où $0 \leq i \leq n$ où cette dernière est appliquée sur D_{i-1}.

Le coût amorti pour i opérations est représenté ainsi :

$$\sum_{i=1}^{n} ca(i) = \sum_{i=1}^{n} cr(i) + \Phi(D_i) - \Phi(D_{i-1}) = \sum_{i=1}^{n} cr(i) + \Phi(D_n) - \Phi(D_0)$$

La fonction de potentiel n'est pas unique, mais il faut que $\Phi(D_n) \geq \Phi(D_0)$ pour que le coût amorti total soit une borne supérieur pour le coût réel.

Contexte

Soit une pile noté P avec les opérations suivantes :

- estVide(P) : retourne vrai si P est vide, sinon faux
- taille(P) : retourne le nombre d'éléments dans la pile
- empiler(P, x) : met x au sommet de la pile P
- depiler(P) : enlève et renvoie l'élément au sommet de la pile P. Si P est vide, renvoie une erreur.
- multiDepiler(P,k) : Dépile $min(k, taille(P))$ éléments.

Analyse classique

En faisant l'analyse classique, les opérations précédemment citées ont la complexité suivante dans le pire des cas :

- estVide(P) : $\Theta(1)$
- taille(P) : $\Theta(n)$ où n est le nombre d'éléments. On peut avoir $\Theta(1)$ si on "track" le nombre d'éléments.
- empiler(P, x) : $\Theta(1)$
- depiler(P) : $\Theta(1)$
- multiDepiler(P,k) : $\Theta(p)$ où $p = min(k, Taille(P))$. Dû que $p \leq n$, la complexité est $\Theta(n)$.

Donc, si on fait n opérations, les opérations les plus coûteuses seront en $\Theta(n^2)$! ! Est-ce vraiment vrai ?

Analyse de l'agrégat

Initialement, la pile est vide. Ce qu'on remarque c'est que la taille est proportiennelle au nombre d'insertions faites. De plus, on ne peut pas dépiler plus que le nombre d'empilements.

- #empilement : nombre d'empilement
- #dépilement : nombre de dépilement

#dépilement $\leq$ #empilement $\leq n$. Donc, n'importe quel opération s'exécutera en $\Theta(n)$ pour n opérations. $ca = \frac{T(n)}{n} = 1$. La complexité est $\Theta(1)$ amortie.

Méthode comptable

Posons que le coût amortie $ca(i)$ pour empiler vaut 2 crédits et le reste valent 0.

C'est à dire on paye un crédit et on garde un crédit.

Intuition

Disons que quand $n = 5$, on fait un dépilement et pour $n = 8$ on fait un multidépilement de 2 et 9 on dépile tout.

n	1	2	3	4	5	6	7	8	9
$\sum_{i=1}^{n} ca(i) - cr(i)$	1	2	3	4	3	4	5	3	0

Pour n opérations, le coût amorti total serait de $2n \in \Theta(n)$. Donc, le coût de chaque opération est de $\frac{2n}{n} = 2 \in \Theta(1)$.

Résumé

Opération	coût réel	coût amortie
Insertion	1	2
Dépilement	1	0
Multidépilement	min(k,n)	0

Donc, la complexité est belle et bien $\Theta(1)$ amortie.

Méthode du potentiel

Posant $\Phi(D_i) = p$ où p représente le nombre d'éléments dans la pile.
Empiler :

$$\begin{aligned} ca(i) &= 1 + p - (p-1) \\ &= 1 + 1 = 2 \end{aligned}$$

Depiler :

$$\begin{aligned} ca(i) &= 1 + (p-1) - p \\ &= 1 - 1 = 0 \end{aligned}$$

MultiDepiler :

$$\begin{aligned} ca(i) &= k + (p-k) - p \\ &= k - k = 0 \end{aligned}$$

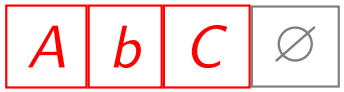
Pour chaque opération, si on effectue une séquence de n opérations, le coût amortie total est inférieur ou égal à $2n$. Donc, chaque opération est effectué en $\Theta(1)$ amortie.

Contexte

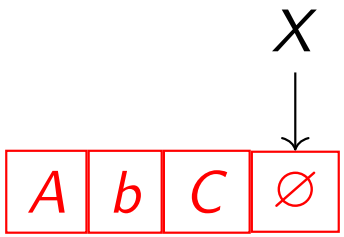
Comparé à son homologue statique, si le tableau est plein et qu'il faut insérer un nouvel élément, celu-ci augmente sa taille pour pouvoir insérer le nouvel élément. C'est sur cette opération qui nous intéressera. On assume que la nouvelle taille qui doit être crée est toujours le double du précédent.

Intuitivement

Supposons qu'on veut insérer un élément nommé X dans le tableau suivant :

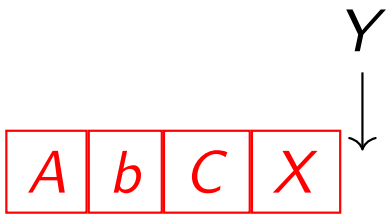


Alors le coût d'insertion sera de 1, car il reste une place dans le tableau :

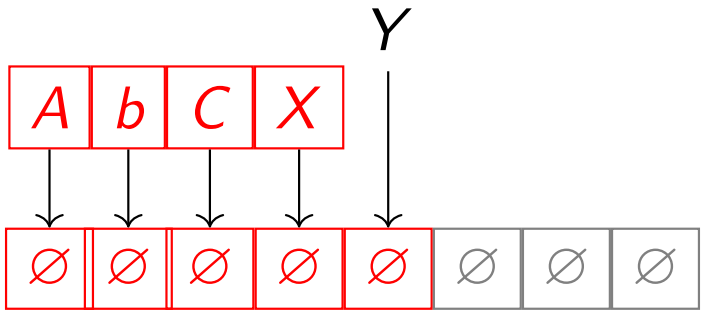


Qu'est ce que si passe si on insére un nouvel élément Y dans le tableau après insertion de X ?

Intuitivement



Solution : crée un tableau plus grand que le précédent, copier les éléments de l'ancien tableau dans le nouveau et ensuite mettre le nouvel élément dans le nouveau.



Cela nous a couté 5 insertions (4 pour transférer les anciens dans le nouveau tableau et 1 pour insérer le nouvel élément) pour ce cas-ci.

Analyse classique

Dans le pire des cas, on a les complexités suivantes :

Insertion à la fin	$\Theta(n)$
Insertion au début/milieu	$\Theta(n^2)$

Est-ce vraiment réaliste ?

Méthode de l'agrégat

Ce qu'on peut remarquer, c'est que l'insertion vaut bel et bien 1, mais le réajustement de taille et son insertion se produit seulement lorsqu'on insére un élément, alors que le nombre d'éléments déjà présent dans le tableau est complet qui est équivalent à une puissance de 2.
Donc, le coût total peut être formulé ainsi si $n > 1$:

$$\begin{aligned} T(n) &= n + \sum_{i=0}^{\lfloor \log_2(n-1) \rfloor} 2^i \\ &= n + \left(\frac{1 - 2^{\lfloor \log_2(n-1) \rfloor + 1}}{1 - 2}\right) \\ &= n + (2(n-1) - 1) = 3n - 3 \in \Theta(n) \end{aligned}$$

Or, $\frac{T(n)}{n} = 1$ et cela donne que la complexité est $\Theta(1)$ amortie.

Méthode du comptable

Quel coût peut-ont donner pour l'insertion ?

$$cr(i) = \begin{cases} 2^{\log_2(i-1)} + 1 & \text{si i - 1 est une puissance de 2} \\ 1 & \text{sinon} \end{cases}$$

On peut écnomiser des crédits losque qu'il faut pas faire de redimenssionnement, mais on paye combien ?

Tentative 1

Disons qu'on décide que $ca(i) = 2$ c'est à dire on paye 1 et on garde un crédit, voyons l'évolution des coûts :

n	1	2	3	4	5	6	7	8	9
$\sum_{i=1}^{n} ca(i) - cr(i)$	1	1	0	1	-2	-1	0	1	-6

Sale inflation je te haie ! ! ! >: (

Tentative 2

Disons qu'on décide que $ca(i) = 3$ c'est à dire on paye 1 et on garde 2 crédits, voyons l'évolution des coûts :

n	1	2	3	4	5	6	7	8	9
$\sum_{i=1}^{n} ca(i) - cr(i)$	2	4	5	7	6	8	10	12	7

En tendant vers n, on obtient $3n$ pour le redimenssionnement et $\frac{3n}{n} = 3 \in \Theta(1)$. Donc, le redimenssionnant d'un tableau se fait en $\Theta(1)$ amortie.

Méthode du potentiel

Posons que $\Phi(D_i) = 2i - 2^{\lceil \log i \rceil}$ qui représente le "crédit" restant.

Pour évaluer la valeur $\Phi(D_i) - \Phi(D_{i-1})$, on doit l'évaluer dans deux cas :

1. Quand $i - 1$ est une puissance de 2

Méthode du potentiel

Posons que $\Phi(D_i) = 2i - 2^{\lceil \log i \rceil}$ qui représente le "crédit" restant.

Pour évaluer la valeur $\Phi(D_i) - \Phi(D_{i-1})$, on doit l'évaluer dans deux cas :

1. Quand $i - 1$ est une puissance de 2
2. Quand $i - 1$ n'est pas une puissance de 2

Cas 1

Si $i-1$ est une puissance de 2, on a :

$$\begin{aligned}\Phi(D_i) - \Phi(D_{i-1}) &= 2i - 2^{\lceil \log i \rceil} - (2(i-1) - 2^{\lceil \log(i-1) \rceil}) \\ &= 2i - 2 \times (2^{\lceil \log(i-1) \rceil}) - (2i - 2 - 2^{\lceil \log(i-1) \rceil}) \\ &= 2i - 2 \times (2^{\lceil \log(i-1) \rceil}) - 2i + 2 + 2^{\lceil \log(i-1) \rceil}) \\ &= -2^{\lceil \log(i-1) \rceil} + 2\end{aligned}$$

Et donc on a :

$$\begin{aligned}ca(i) &= cr(i) + \Phi(D_i) - \Phi(D_{i-1}) \\ &= 1 + 2^{\lceil \log(i-1) \rceil} - 2^{\lceil \log(i-1) \rceil} + 2 \\ &= 1 + 2 = 3 \in \Theta(1)\end{aligned}$$

Quand est-t'il du cas 2 ?

Cas 2

Si $i-1$ n'est pas une puissance de 2, alors $2^{\lceil \log i \rceil} = 2^{\lceil \log(i-1) \rceil}$ et donc :

$$\begin{aligned}\Phi(D_i) - \Phi(D_{i-1}) &= 2i - 2^{\lceil \log i \rceil} - (2(i-1) - 2^{\lceil \log i \rceil}) \\ &= 2i - 2^{\lceil \log i \rceil} - (2i - 2 - 2^{\lceil \log i \rceil}) \\ &= 2\end{aligned}$$

Donc, on obtient :

$$\begin{aligned}ca(i) &= cr(i) + \Phi(D_i) - \Phi(D_{i-1}) \\ &= 1 + 2 = 3 \in \Theta(1)\end{aligned}$$

Donc, la complexité pour redimenssionner un tableau dynamique est bel et bien $\Theta(1)$ amortie.

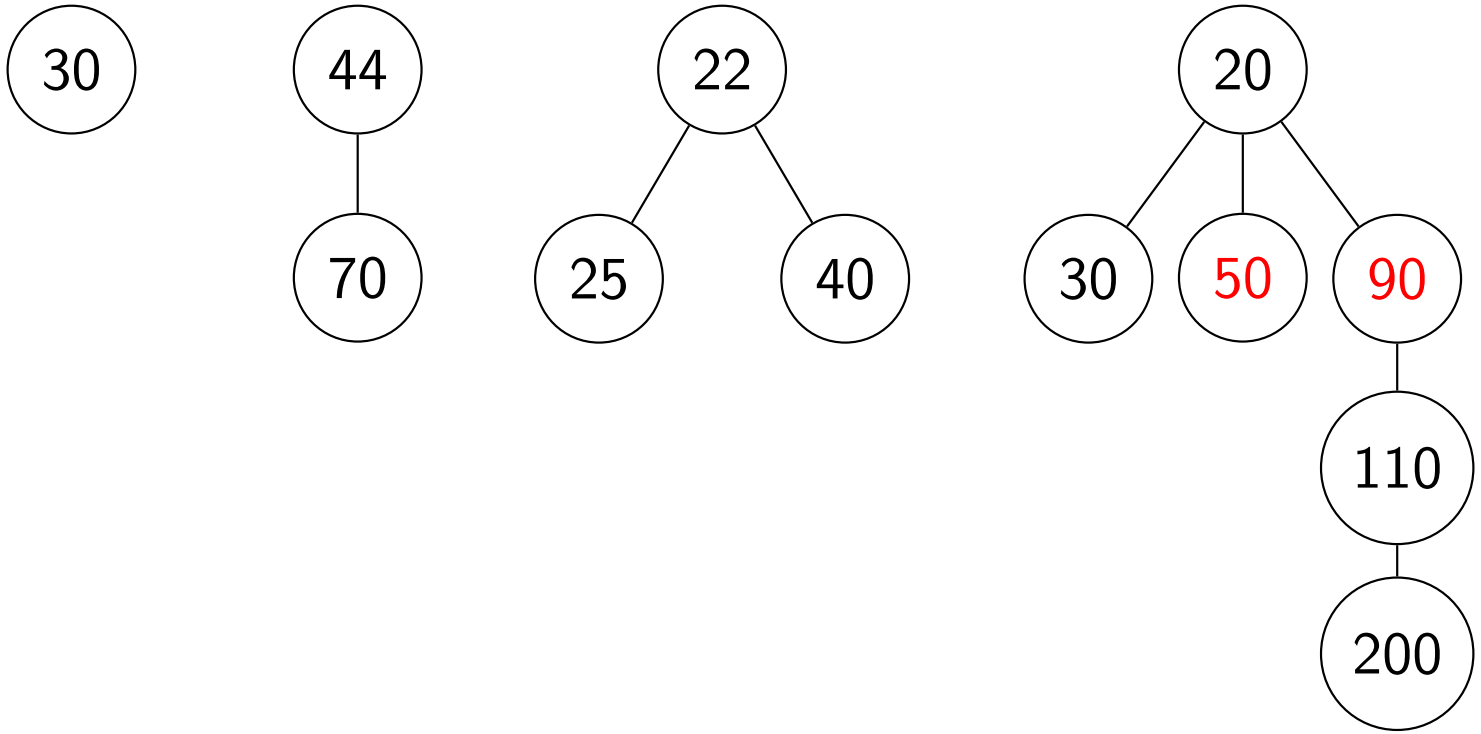
Différence

Comparé à un monceau binaire, ce monceau est un ensemble d'arbres (Forêt). De plus, chaque noeud contient un degré (rang) qui représente le nombre d'enfants que la noeud possède.

Pourquoi monceau de "Fibonnacci" ? On verra plus tard pourquoi. :)

Example de monceau de Fibonnacci

Voici un exemple de monceau de Fibonnacci à 12 clés :



Les noeuds en rouges sont des noeuds marqués (servira pour la réduction de priorité).

Opérations

Les opérations qu'on étudira seront :

- Obtenir le minimum/maximum

Opérations

Les opérations qu'on étudira seront :

- Obtenir le minimum/maximum
- Insérer une nouvelle clé

Opérations

Les opérations qu'on étudira seront :

- Obtenir le minimum/maximum
- Insérer une nouvelle clé
- Enlever le minimum/maximum

Opérations

Les opérations qu'on étudira seront :

- Obtenir le minimum/maximum
- Insérer une nouvelle clé
- Enlever le minimum/maximum
- Réduction/augmentation de priorité d'une clé

Opérations

Les opérations qu'on étudira seront :

- Obtenir le minimum/maximum
- Insérer une nouvelle clé
- Enlever le minimum/maximum
- Réduction/augmentation de priorité d'une clé

Le changement de priorité d'une clé est l'opération la plus intéressante.

Obtenir le minimum

Pour trouver le minimum il y a deux façons :

- Parcourir tout les arbres et prendre la valeur minimale ou maximale en comparant toutes les racines. (Naive)

Quelle est la complexité des deux approches ?

Naive : $\mathcal{O}(\log n)$.

L'optimisé : $\mathcal{O}(1)$.

Obtenir le minimum

Pour trouver le minimum il y a deux façons :

- Parcourir tout les arbres et prendre la valeur minimale ou maximale en comparant toutes les racines. (Naive)
- Garder toujours un suivi du minimum de la racine minimale/maximale.

Quelle est la complexité des deux approches ?

Naive : $\mathcal{O}(\log n)$.

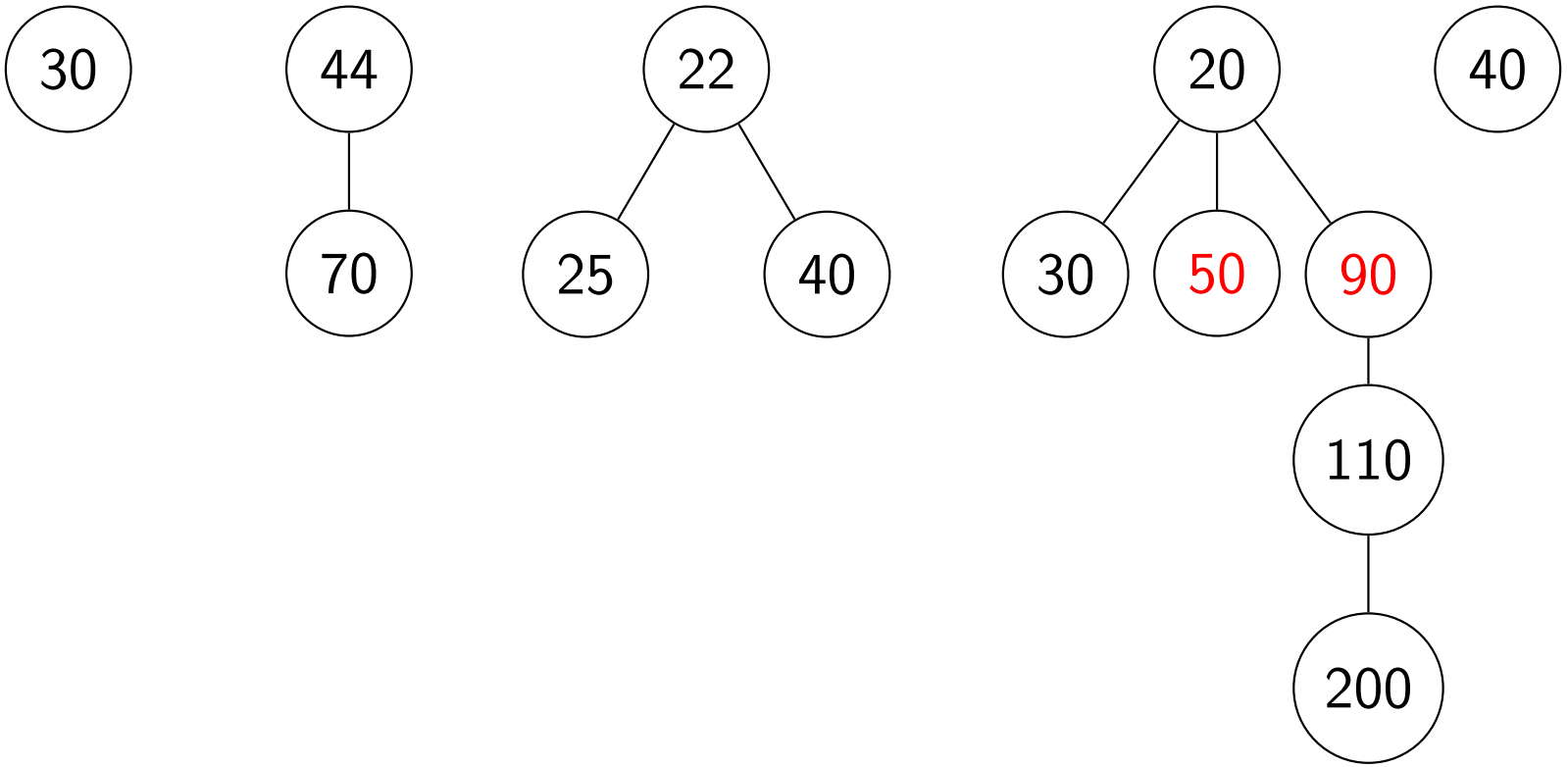
L'optimisé : $\mathcal{O}(1)$.

Insérer une nouvelle clé

Pour insérer une nouvelle clé il faut juste crée une nouvelle racine et comparer avec l'ancien minimum et le mettre à jour si nécessaire.
Complexité : complexité pour obtenir le minimum ($\mathcal{O}(1)$).

Exemple

Supposons qu'on veut insérer l'élément 40 dans notre monceau, on aura le monceau suivant après insertion :



Supprimer le minimum

Cette opération est divisée en trois parties :

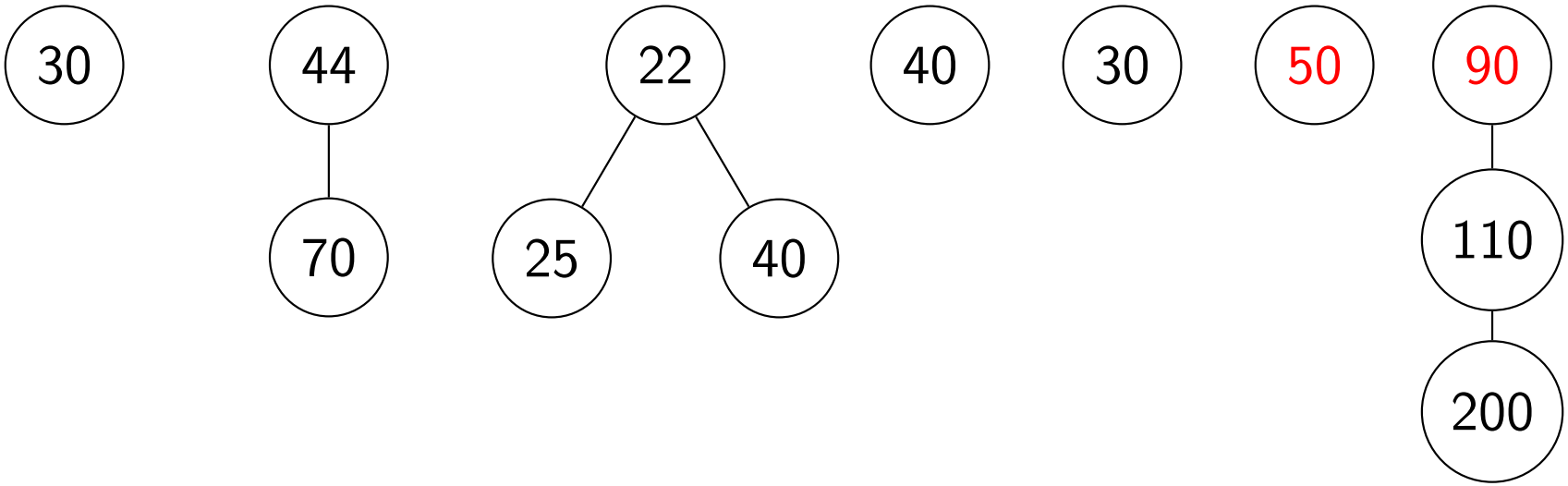
1. Suppression du minimum et déplacement d'enfants.
2. Fusion des noeuds de même degré
3. Trouver le minimum.

Act 1

La première chose à faire est que si le minimum a des enfants, il faut d'abord retirer les enfants et les mettre à la fin du monceau en tant que racines et ensuite on peut enlever le minimum.

Exemple

Supossons que l'on veut retirer le minimum de notre file de tout à l'heure, on obtient le monceau dans l'état suivant :



Act 2

Créer un tableau d'une taille $p + 1$ où p est le degrée maximale d'un arbre pour le nombre de noeuds courant.

Case [i] tableau : degré d'une racine dans le monceau.

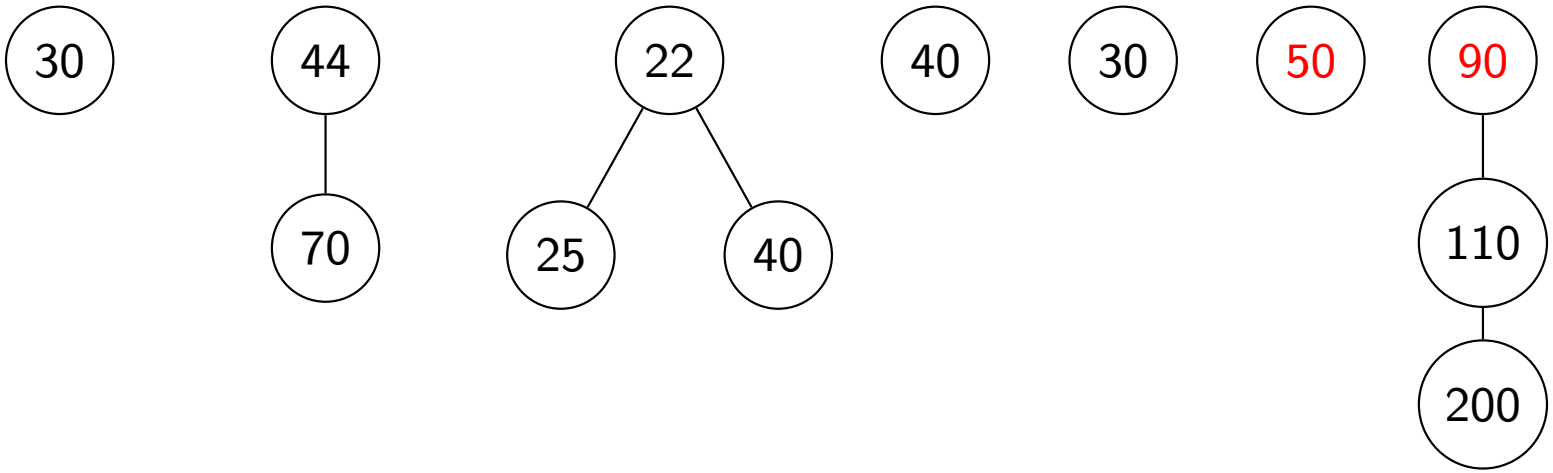
Si deux monceaux peuvent allés dans la case, alors la plus grande valeur des deux racines devient un enfant de l'autre racine et on ensuite on le déplace dans la prochaine case du tableau jusqu'à que la case ne soit pas occupé.

On répete le processus jusqu'à avoir traiter toutes les racines.

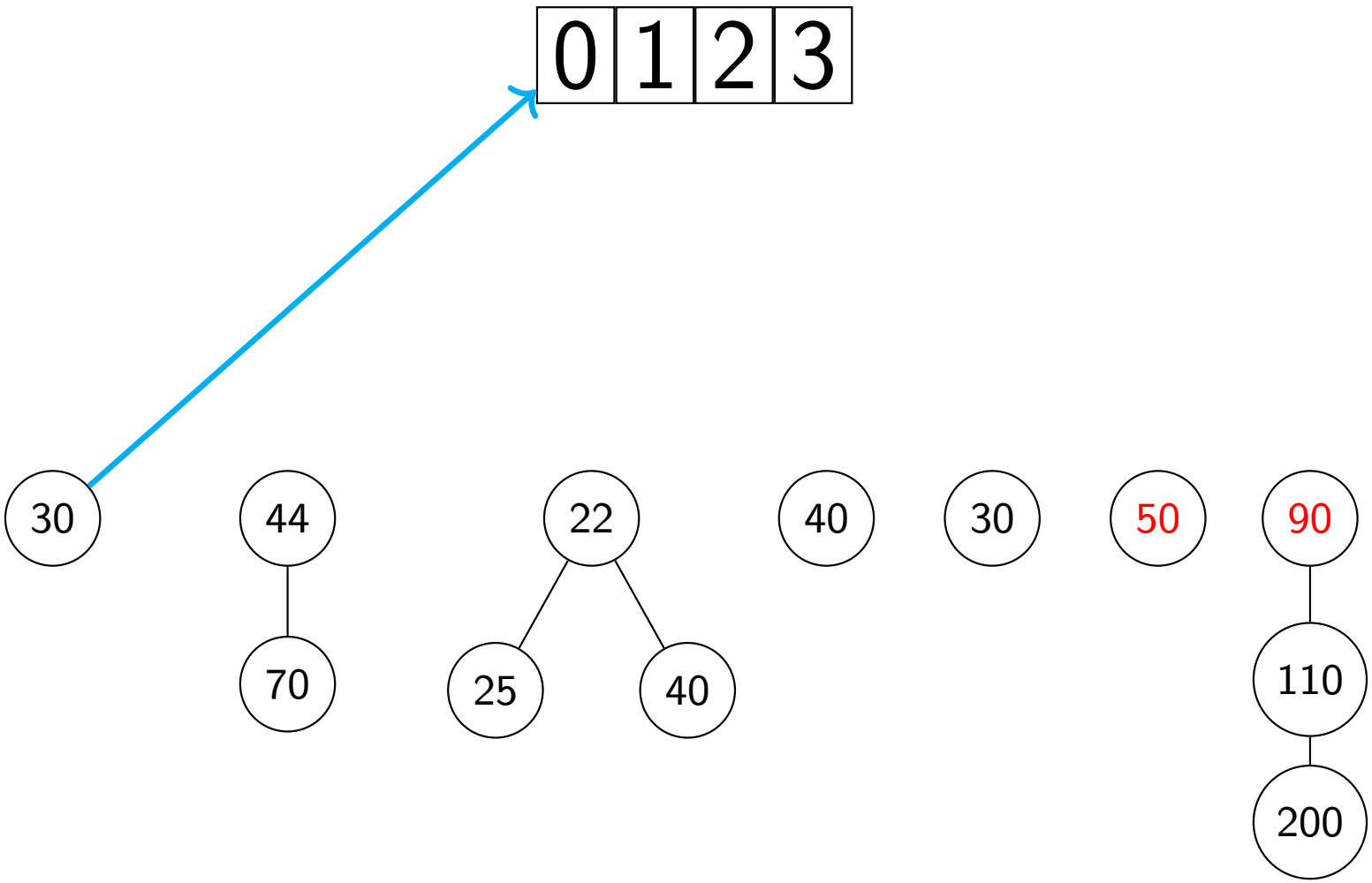
Exemple

En reprenant notre exemple de tout à l'heure, notre tableau doit être de taille 4, ce qui donne le tableau suivant :

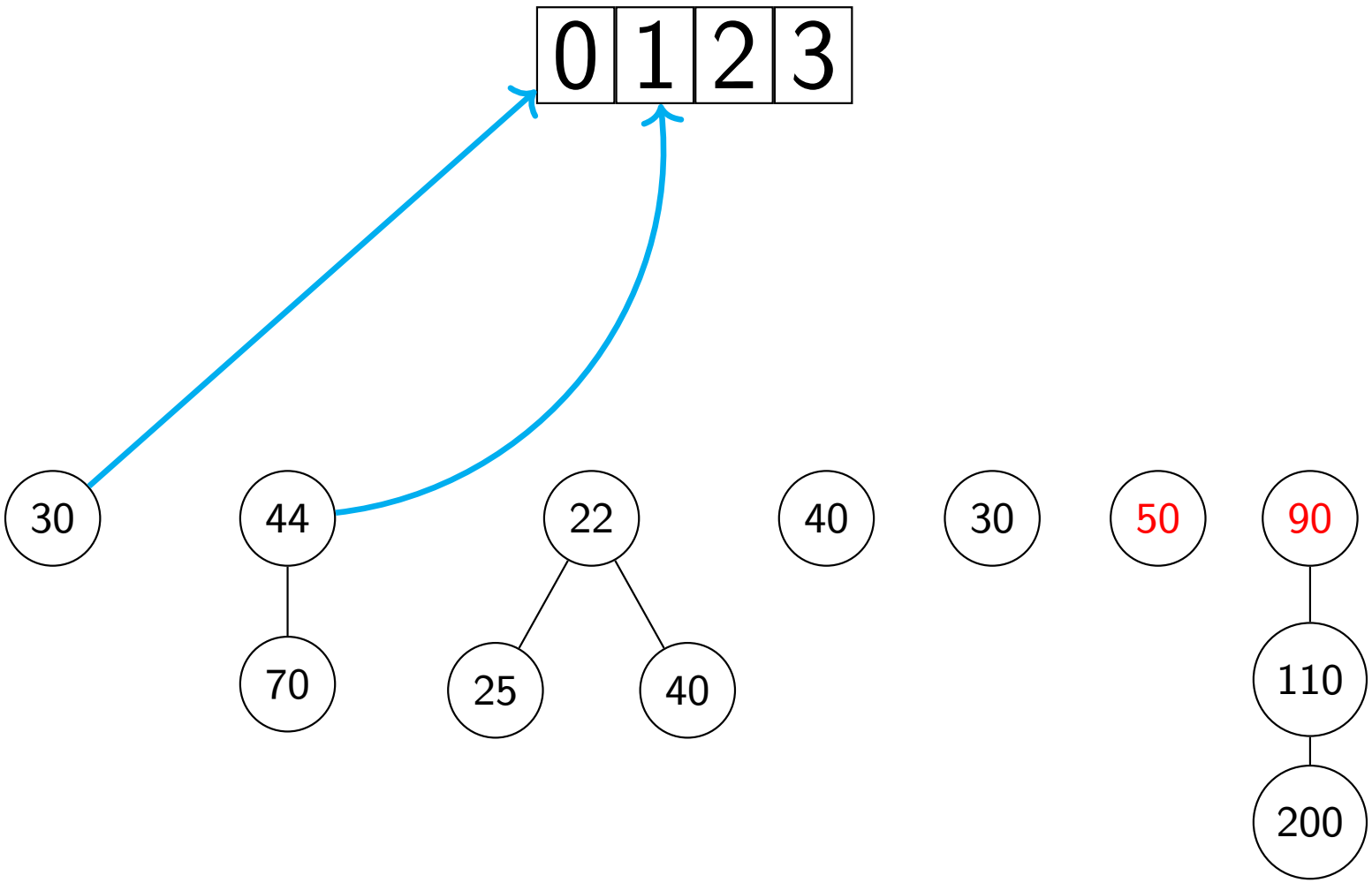
0	1	2	3



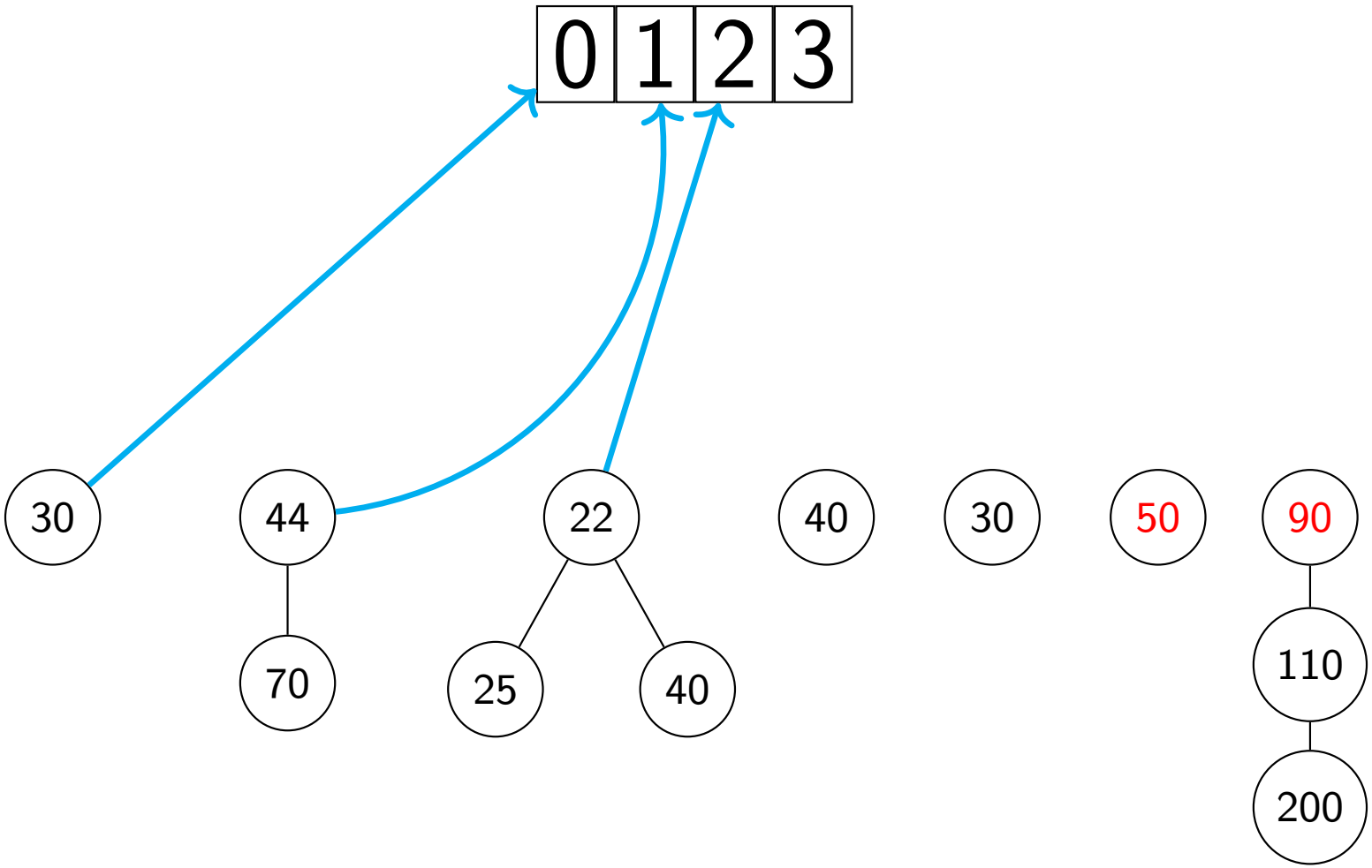
Racine 1



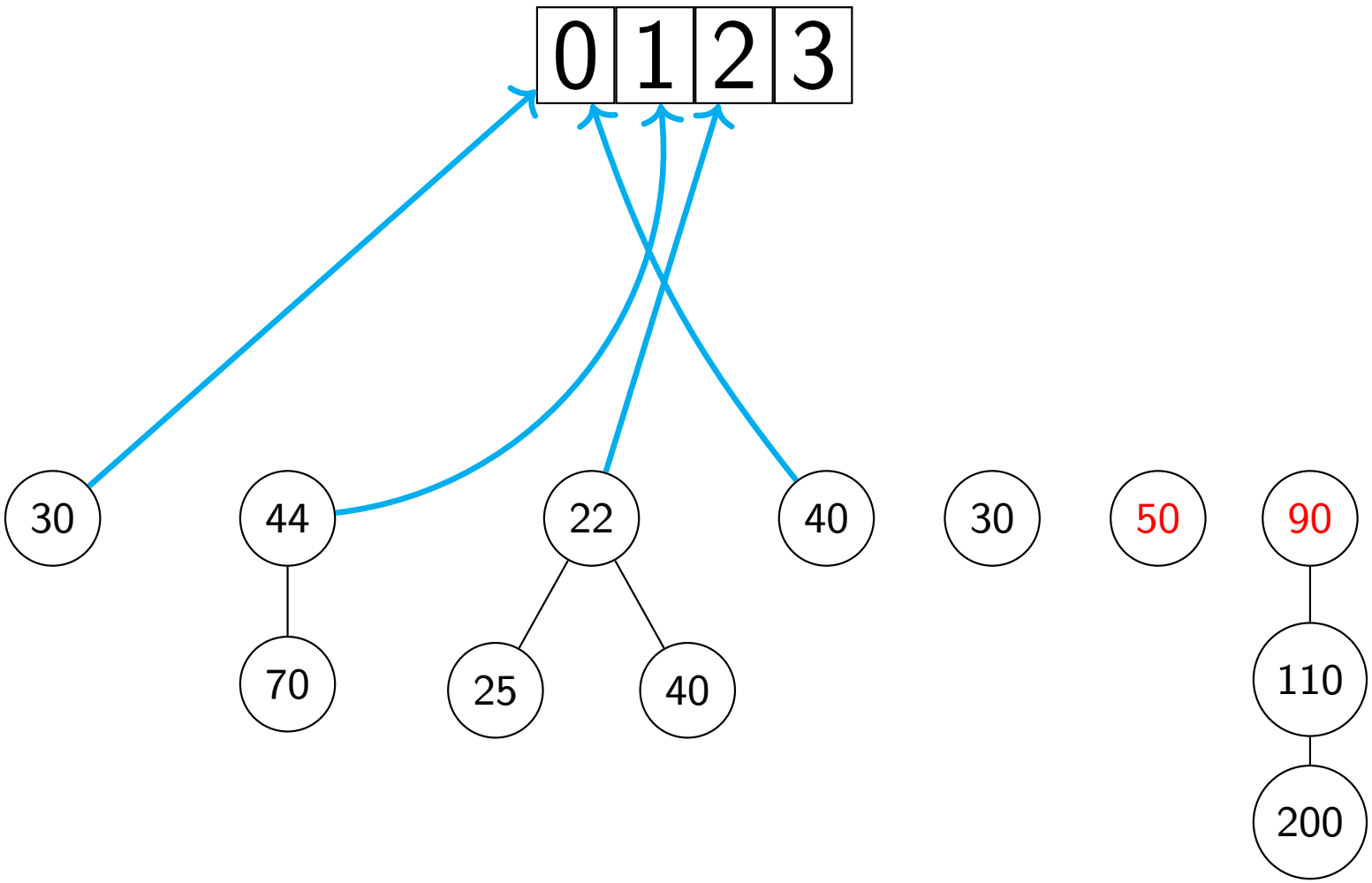
Racine 2



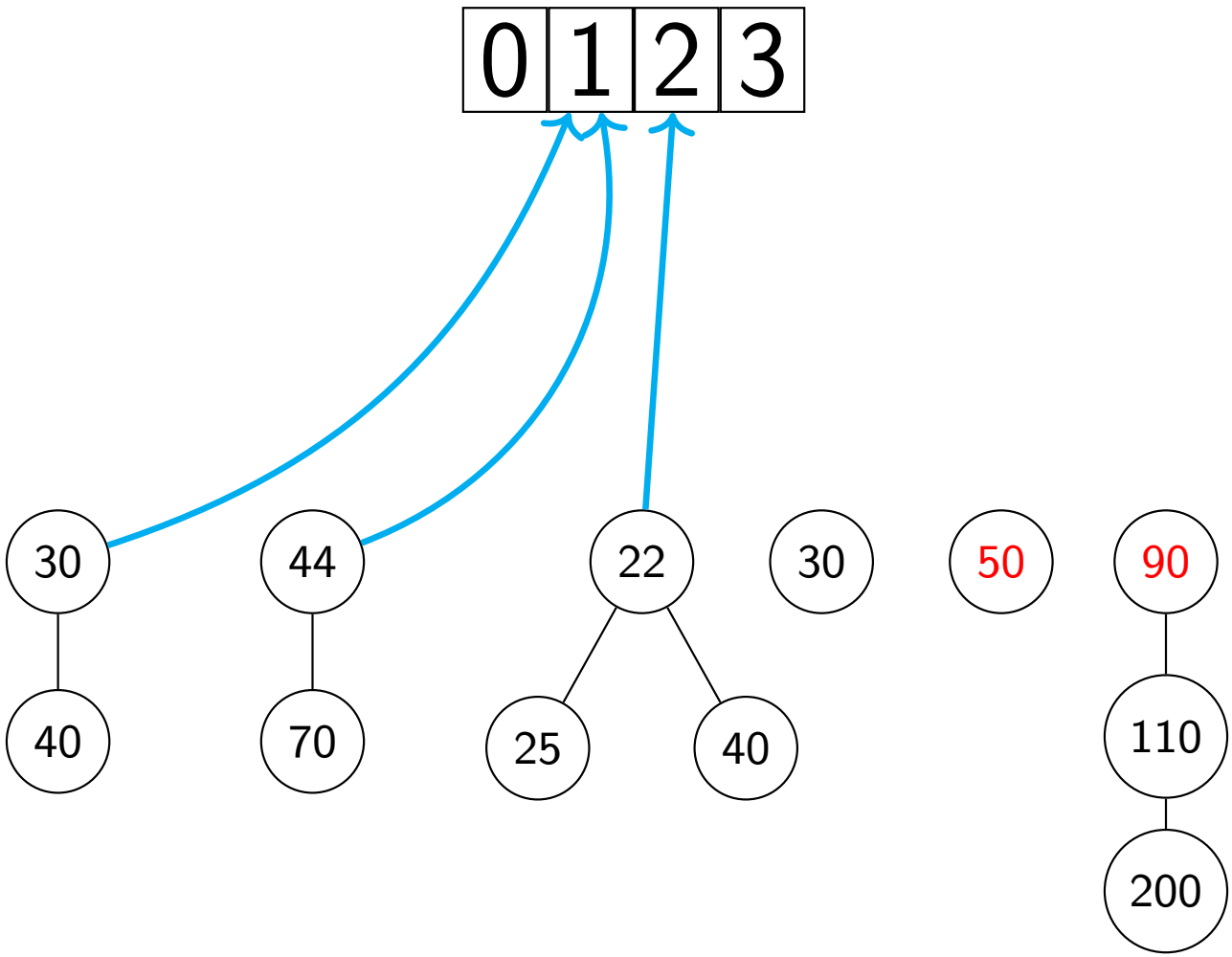
Racine 3



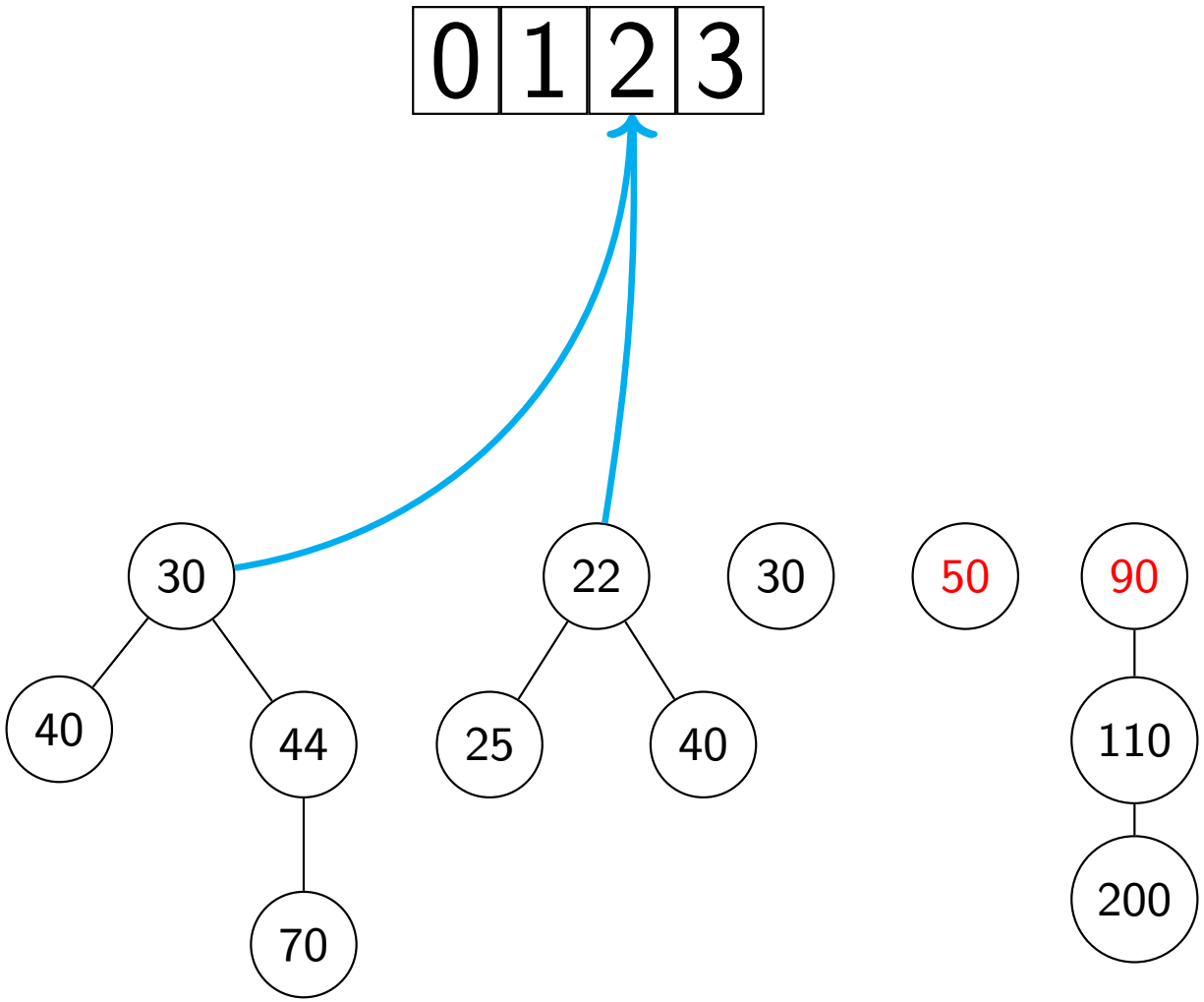
Racine 4



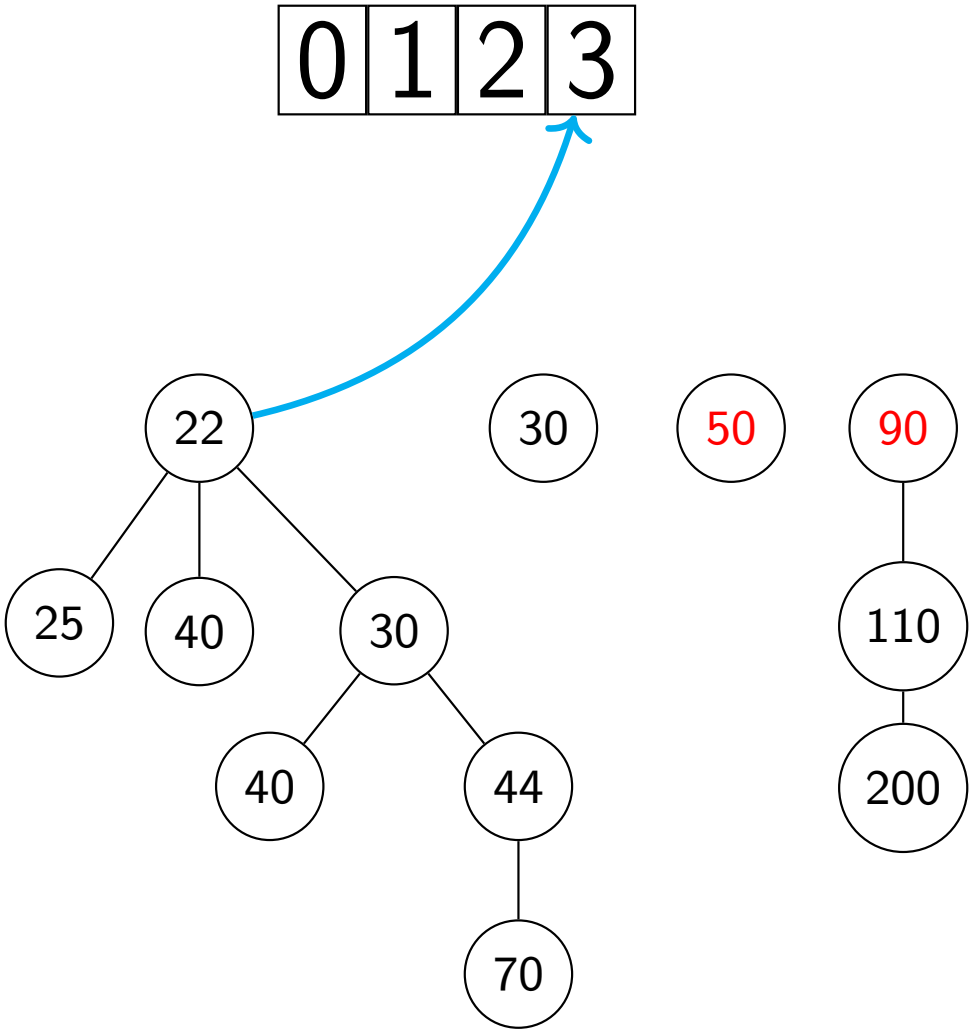
Fusion 1



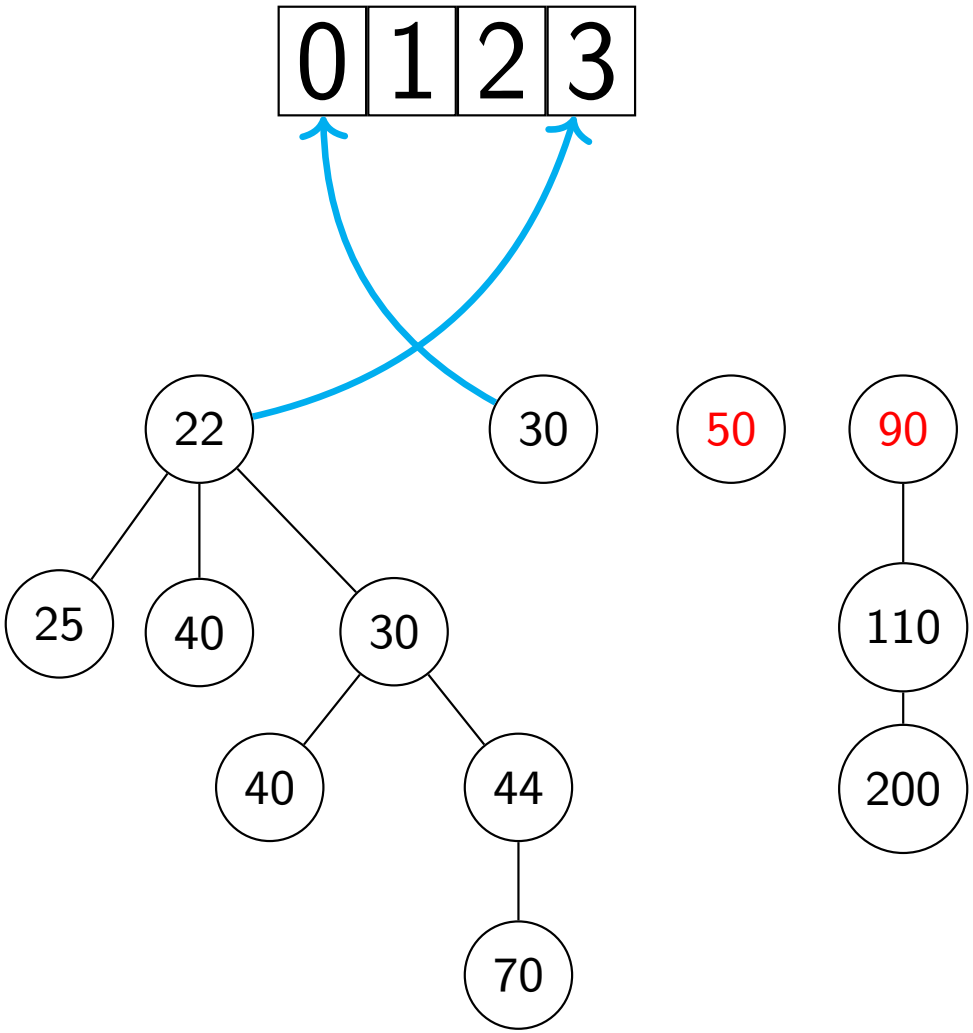
Fusion 2



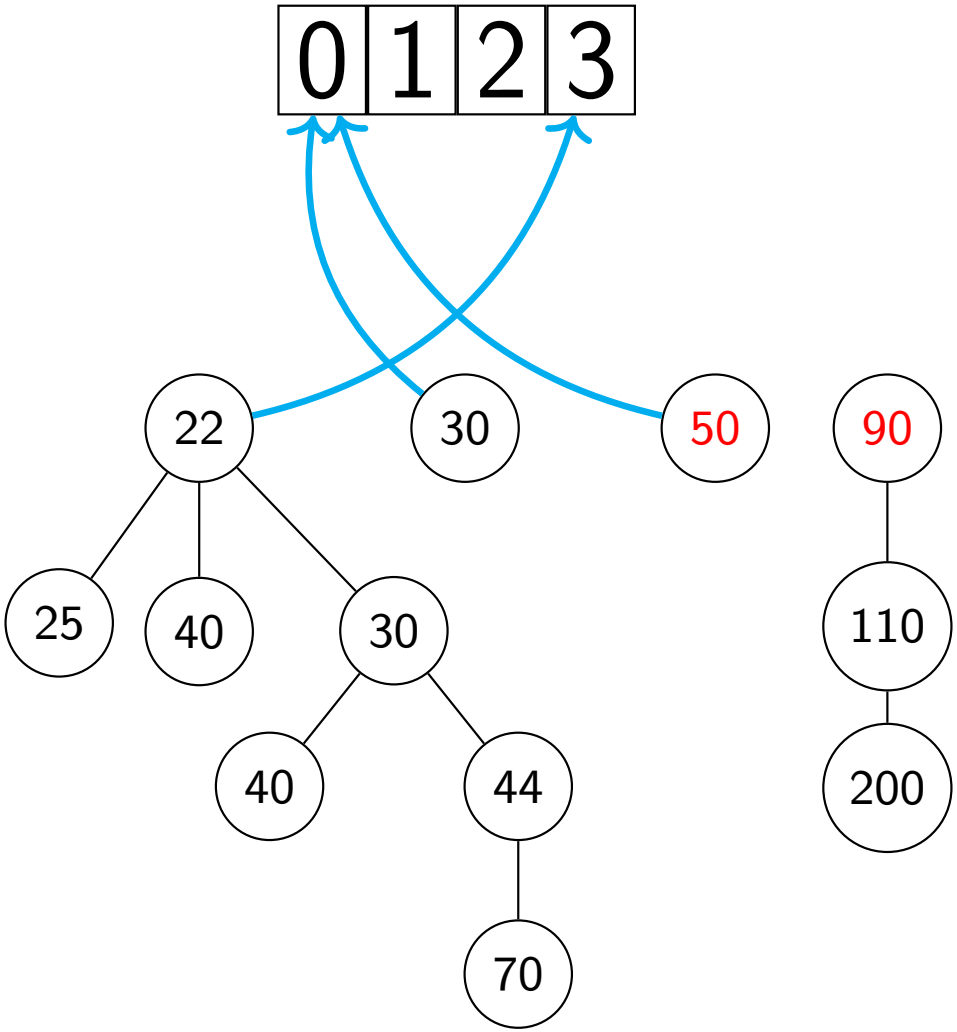
Fusion 3



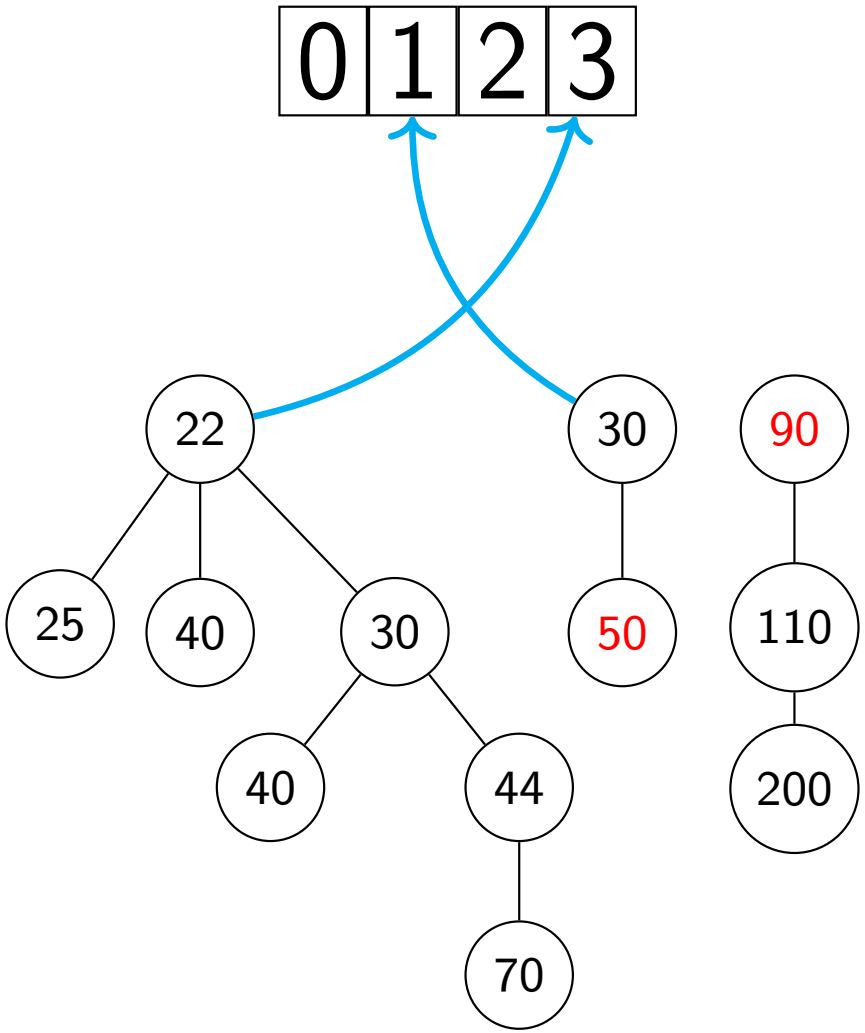
Racine 5



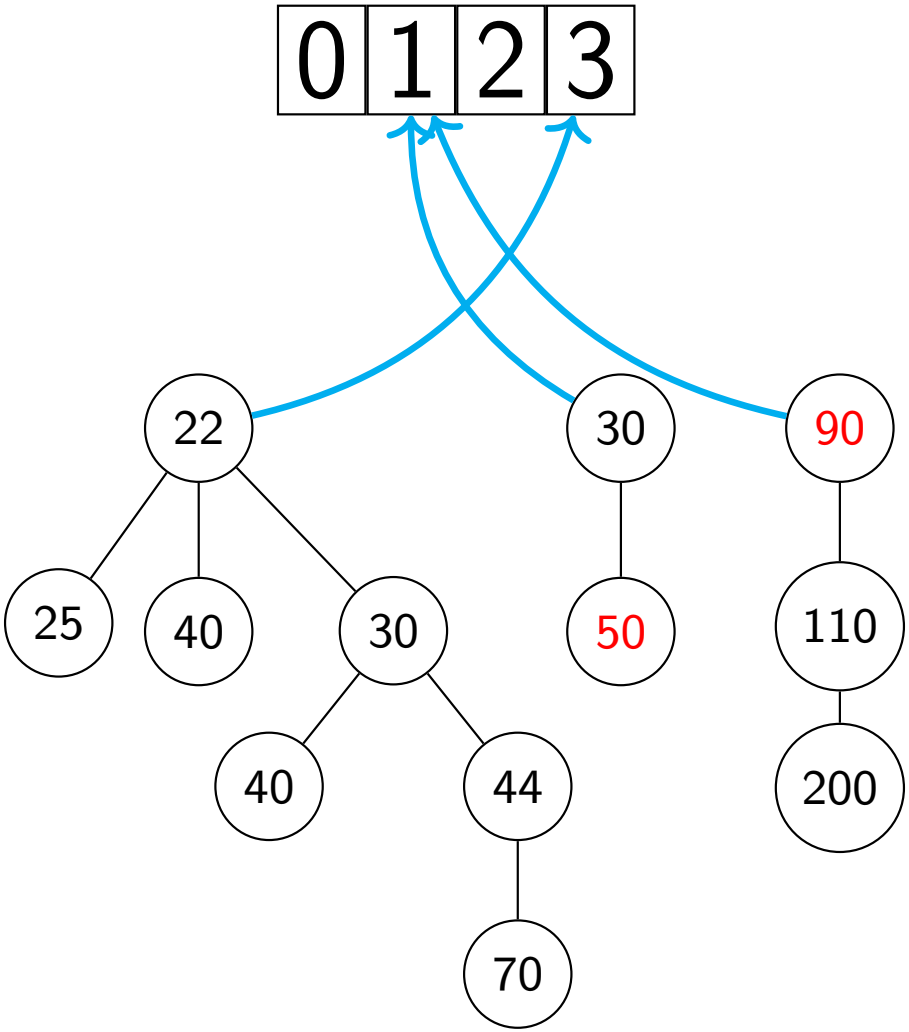
Racine 6



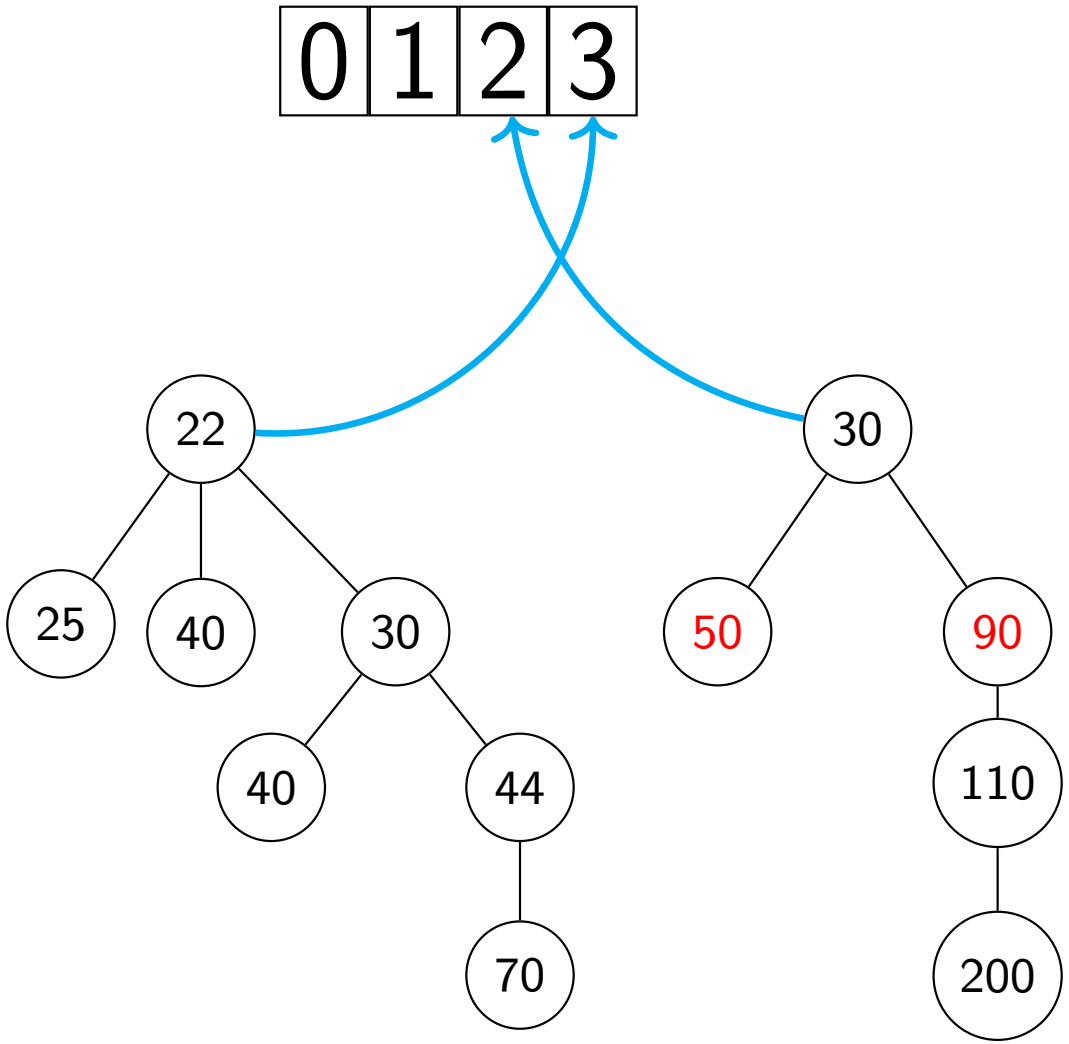
Fusion



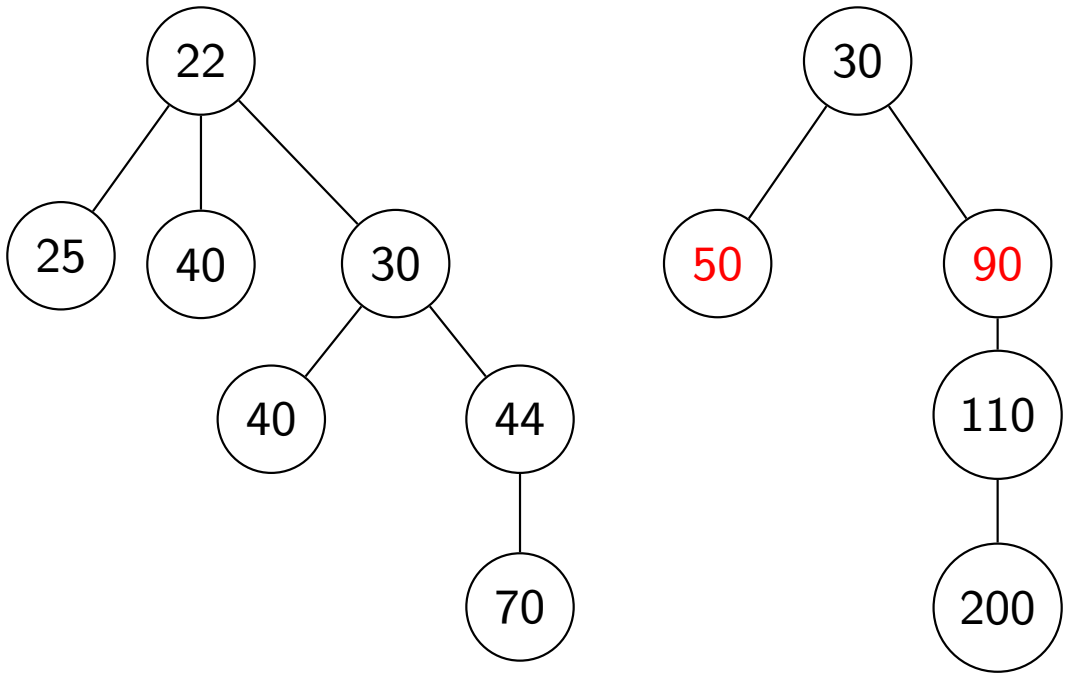
Racine 7



Fusion



État final



Et voilà ! !

Act 3

Dans le monceau de Fibonnaci, après la deuxième étape, on cherche le minimum.

Coût

Le coût total de l'opération peut être représenté de la manière suivante :

1. Étape 1 : Θ(degrée du minimum)
2. Étape 2 : Θ(degrée maximale + #arbres)
3. Étape 3 : Θ(degrée maximale)

Ok, mais que vaut le degrée maximale ? Que se passe t'il si on enlève le minimum avec juste des racines de degré 0 ?

Intuition

Ce qu'on remarque c'est qu'on fusionne deux racines parce qu'ils ont le même degrée. Une fois la fusion faite, la nouvelle racine augmente son degré de 1 et la somme de celle-ci des éléments des deux racines fusionnées est de $\leq 2^d$ où d est le degré des deux racines qui ont été fusionnées.

- degrée 0 : 1

Intuition

Ce qu'on remarque c'est qu'on fusionne deux racines parce qu'ils ont le même degrée. Une fois la fusion faite, la nouvelle racine augmente son degré de 1 et la somme de celle-ci des éléments des deux racines fusionnées est de $\leq 2^d$ où d est le degré des deux racines qui ont été fusionnées.

- degrée 0 : 1
- degrée 1 : 2

Intuition

Ce qu'on remarque c'est qu'on fusionne deux racines parce qu'ils ont le même degrée. Une fois la fusion faite, la nouvelle racine augmente son degré de 1 et la somme de celle-ci des éléments des deux racines fusionnées est de $\leq 2^d$ où d est le degré des deux racines qui ont été fusionnées.

- degrée 0 : 1
- degrée 1 : 2
- degrée 2 : 4

Intuition

Ce qu'on remarque c'est qu'on fusionne deux racines parce qu'ils ont le même degrée. Une fois la fusion faite, la nouvelle racine augmente son degré de 1 et la somme de celle-ci des éléments des deux racines fusionnées est de $\leq 2^d$ où d est le degré des deux racines qui ont été fusionnées.

- degrée 0 : 1
- degrée 1 : 2
- degrée 2 : 4
- degrée 3 : 8

Intuition

Ce qu'on remarque c'est qu'on fusionne deux racines parce qu'ils ont le même degrée. Une fois la fusion faite, la nouvelle racine augmente son degré de 1 et la somme de celle-ci des éléments des deux racines fusionnées est de $\leq 2^d$ où d est le degré des deux racines qui ont été fusionnées.

- degrée 0 : 1
- degrée 1 : 2
- degrée 2 : 4
- degrée 3 : 8
- degrée 4 : 16

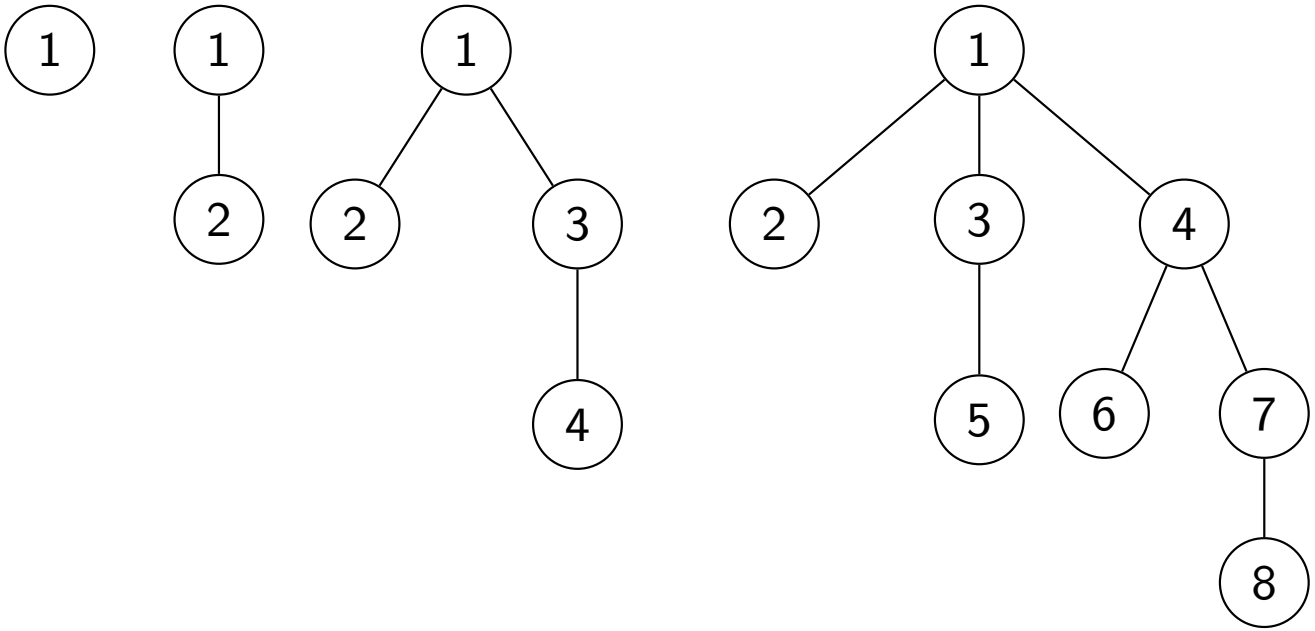
Intuition

Ce qu'on remarque c'est qu'on fusionne deux racines parce qu'ils ont le même degrée. Une fois la fusion faite, la nouvelle racine augmente son degré de 1 et la somme de celle-ci des éléments des deux racines fusionnées est de $\leq 2^d$ où d est le degré des deux racines qui ont été fusionnées.

- degrée 0 : 1
- degrée 1 : 2
- degrée 2 : 4
- degrée 3 : 8
- degrée 4 : 16

Note : on suppose qu'on fait pas de réduction de clés.
Pas convaincu ?

Intuition visuelle



Remarque : ces arbres sont des arbre binomiaux et ils augmentent de façon exponentielle. Soit d_{max} le degré maximale d'un arbre à n noeuds, alors $\log_2 n = d_{max}$.

degrée maximale : log #nombre de clés $\in \mathcal{O}(\log n)$

Modification de priorité de clé

Suppossons qu'on a un accès direct sur une clé (table de hachage), la suppression d'une clé se fait comme suit :

- Changer la clé avec la nouvelle priorité.

Modification de priorité de clé

Suppossons qu'on a un accès direct sur une clé (table de hachage), la suppression d'une clé se fait comme suit :

- Changer la clé avec la nouvelle priorité.
- Si la clé modifié est toujours plus grand que son parent, alors on ne fait rien. Sinon, on doit couper ce noeud (avec ses enfants) et le mettre dans la liste des racines. Mettre à jour le min si nécessaire.

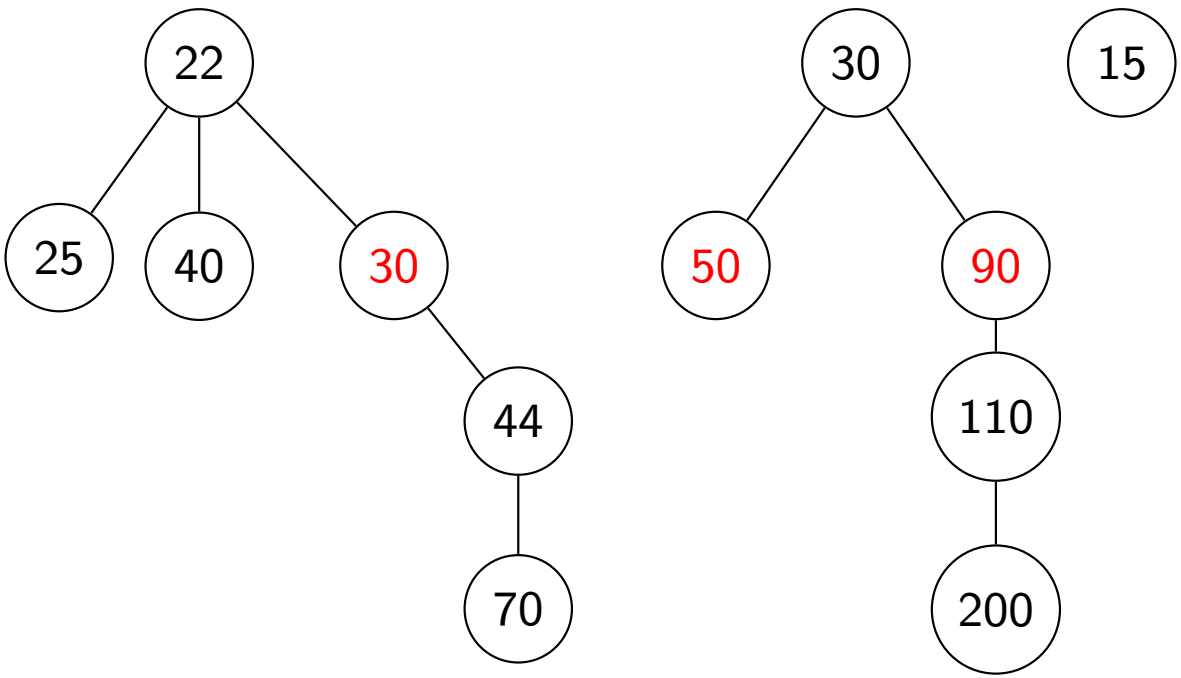
Modification de priorité de clé

Suppossons qu'on a un accès direct sur une clé (table de hachage), la suppression d'une clé se fait comme suit :

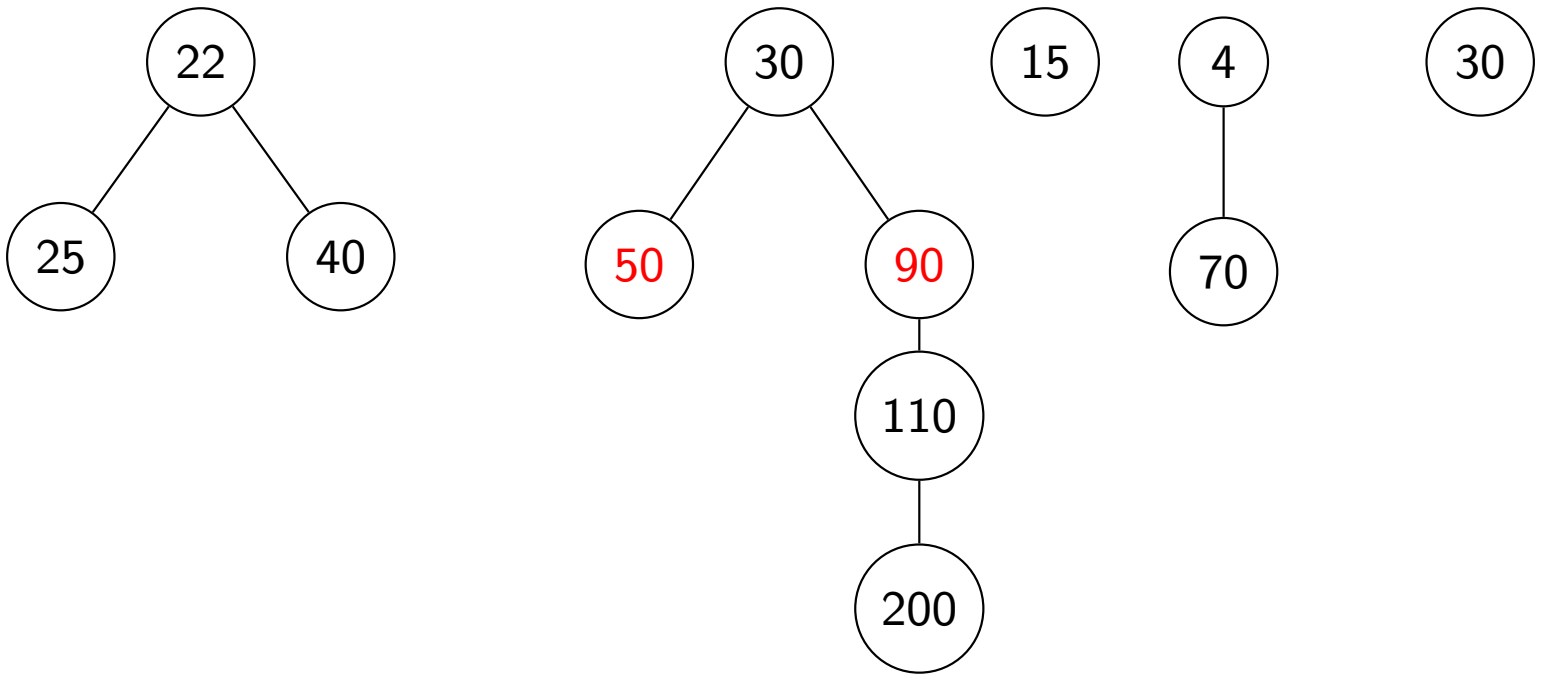
- Changer la clé avec la nouvelle priorité.
- Si la clé modifié est toujours plus grand que son parent, alors on ne fait rien. Sinon, on doit couper ce noeud (avec ses enfants) et le mettre dans la liste des racines. Mettre à jour le min si nécessaire.
- Si le parent a perdu un enfant, alors il est marqué. S'il est déjà marqué, alors à son tour il se fait mettre dans liste des racines et on enlève sa marque. On refait le même processus jusqu'à atteindre la racine de l'arbre modifié ou jusqu'à atteindre un parent non marqué.

Exemple (40 $\rightarrow$ 15)

En reprenant toujours le même arbre après suppression du minimum, réduisons les clé 40 (rattaché à 30) pour 15 et 44 pour 4 :



Réduction clé 44 $\longrightarrow$ 4



Est-ce que les arbres grossit exponentiellement ?

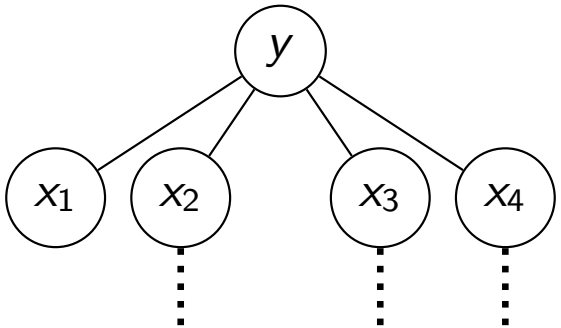
Lors de l'extraction du minimum, on fusionne deux racines,car ils sont de même ordre et la nouvelle racine est du même ordre que les deux racines plus 1.

Notre règle de réduction de clé et qu'un parent peut rester dans l'arbre tant qu'il ne perd pas plus qu'un **enfant**, sinon, il faut enlever le noeud parent de l'arbre.

Dans le cas où le noeud a déjà été marqué, il ne doit pas perdre d'enfants pour pouvoir rester dans l'arbre sinon coupure.

En d'autre termes, cette règle permet de limiter la quantité de noeuds dans l'arbre analoguement avec le degré de la racine de cette arbre.

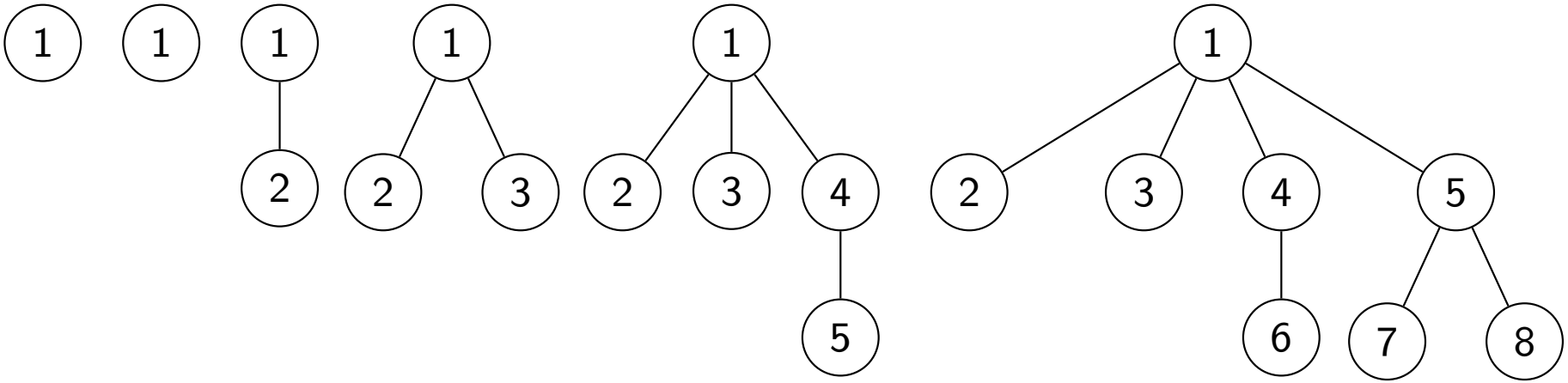
Intuition



Pour que x_4 soit fusionner avec y, y était de degré 3 et x_4 de degré 3. Si x_4 perd un enfant et il n'est pas marqué, alors il sera marqué et son degré descendera de 1. Si x_4 perd un autre enfant et qu'il a déjà été marqué, alors il doit être enlever de cet arbre et son degré est décrémenter de 1 et il x_4 devient une nouvelle racine. De plus, le degré de y est aussi décrémenté de 1 dû que x_4 n'est plus un enfant de y.

Donc, le degré de x_4 en tant que nouvelle racine est au moins nouveau degré $\geq d_4 - 2$. Construisons la taille minimale que les arbres auront avec cette règle.

Minimum



Remarquez-vous la séquence ? Suite de Fibonnacci ! !

Si on fait le rapport entre un nombre de Fibonnacci et son suivant, on obtient le résultat suivant :

$$\lim_{n\to\infty} \frac{F_{n+1}}{F_n} = \varphi = \frac{1+\sqrt{5}}{2} \approx 1,618$$

où F_i est inième terme de la suite de Fibonnacci. Fait intéressant : $\varphi^2 = \varphi + 1$.

Nombre de noeuds dans un arbre

Soit, d le degré de l'arbre d'un monceau et n son nombre de noeuds, alors le nombre de noeud est le suivant :

$$F_{d+2} \leq n$$

Note : on peut démontrer que $n \geq \varphi^d$ ou dans ce cas-ci $F_{d+2} \geq \varphi^d$.

Analyse

Combien de noeuds sont au maximum coupés ? On coupe soit un noeud parce que la modification de clé n'est pas respecté entre la clé modifié ou que le parent a déjà été marqué.

En d'autre termes, le nombre de réduction de clés k engendre au plus $2k$ coupures de noeuds. Donc, $2k$ futures racines.

Analyse

Posons la fonction de potentiel suivante : $\aleph(D_i) = t + 2m$ où t est le nombre d'arbres (racines) dans le monceau et m le nombre de noeuds marqués.

L'insertion :

$$\begin{aligned} ca(i) &= cr(i) + \aleph(D_i) - \aleph(D_{i-1}) \\ &= 1 + (t + 1 + 2m) - (t + 2m) \\ &= 1 + 1 = 2 \in \Theta(1) \end{aligned}$$

Trouver le minimum :

$$\begin{aligned} ca(i) &= cr(i) + \aleph(D_i) - \aleph(D_{i-1}) \\ &= 1 + (t + 2m) - (t + 2m) \\ &= 1 + 0 = 1 \in \Theta(1) \end{aligned}$$

Analyse

Supprimer le minimum :

$$\begin{aligned} ca(i) &= cr(i) + \aleph(D_i) - \aleph(D_{i-1}) \\ &= D(t) + t + (D(t) + 1 + 2m) - (t + 2m) \\ &= 2D(t) + 1 \in \Theta(D(t)) \in \Theta(\log n) \end{aligned}$$

Réduction de clés

$$\begin{aligned} ca(i) &= cr(i) + \aleph(D_i) - \aleph(D_{i-1}) \\ &= c + (t + c + 2(m - c + 2)) - (t + 2m) \\ &= c + (t + c + 2m - 2c + 4) - (t + 2m) \\ &= 4 \in \Theta(1) \end{aligned}$$

Note : $c =$ nombre de coupures.

Conclusion

Donc, voici les complexités en amorties obtenu :

1. Obtenir le minimum/maximum : $\Theta(1)$
2. Insérer une clé : $\Theta(1)$
3. Supprimer le minimum/maximum : $\Theta(\log n)$
4. Réduction/augmentation de priorité d'une clé : $\Theta(1)$