



INFO0501

ALGORITHMIQUE AVANCÉE

COURS 3

STRUCTURES DE DONNÉES DYNAMIQUES
TABLES DE HACHAGE ET ARBRES BINAIRES DE RECHERCHE



UNIVERSITÉ
DE REIMS
CHAMPAGNE-ARDENNE

Pierre Delisle
Département de Mathématiques et Informatique
Septembre 2018

Plan de la séance

- Tables de hachage
- Arbres binaires de recherche
- Arbres rouge et noir
- Bibliographie
 - T. H. Cormen, C. E. Leiserson, R. L. Rivest, C. Stein, "Algorithmique", 3^e édition, Dunod, 2010



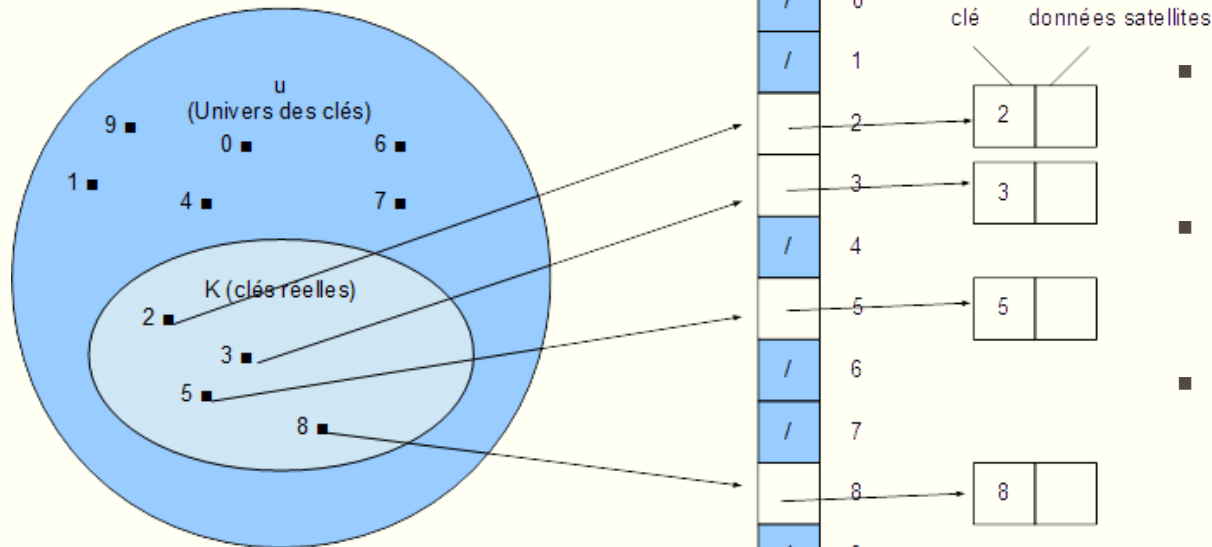
TABLES DE HACHAGE

Table de hachage ?

- Beaucoup d'applications nécessitent des ensembles dynamiques qui ne supportent que
 - INSÉRER, RECHERCHER ET SUPPRIMER
- Implémentation efficace des dictionnaires
 - INSÉRER ET SUPPRIMER $\rightarrow O(1)$
 - RECHERCHER $\rightarrow O(n)$ en pire cas
 - ... mais on peut obtenir un temps moyen de $O(1)$
- Généralisation de la notion de tableau
 - Tableau \rightarrow table à adressage direct

Table à adressage direct

- Efficace lorsque l'univers u des clés est raisonnablement petit
- Chaque élément possède une clé prise dans l'univers $u = \{0, 1, \dots, m - 1\}$
 - où m n'est « pas trop grand »
 - ... et deux éléments ne peuvent avoir la même clé
- $u = \{0, 1, \dots, 9\}$
- $k = \{2, 3, 5, 8\}$



- On utilise un tableau
 - $t[0 .. m - 1]$
 - L'alvéole k pointe sur l'élément ayant pour clé k

▪ RECHERCHER-ADRESSAGE-DIRECT (t, k)

▪ INSÉRER-ADRESSAGE-DIRECT (t, x)

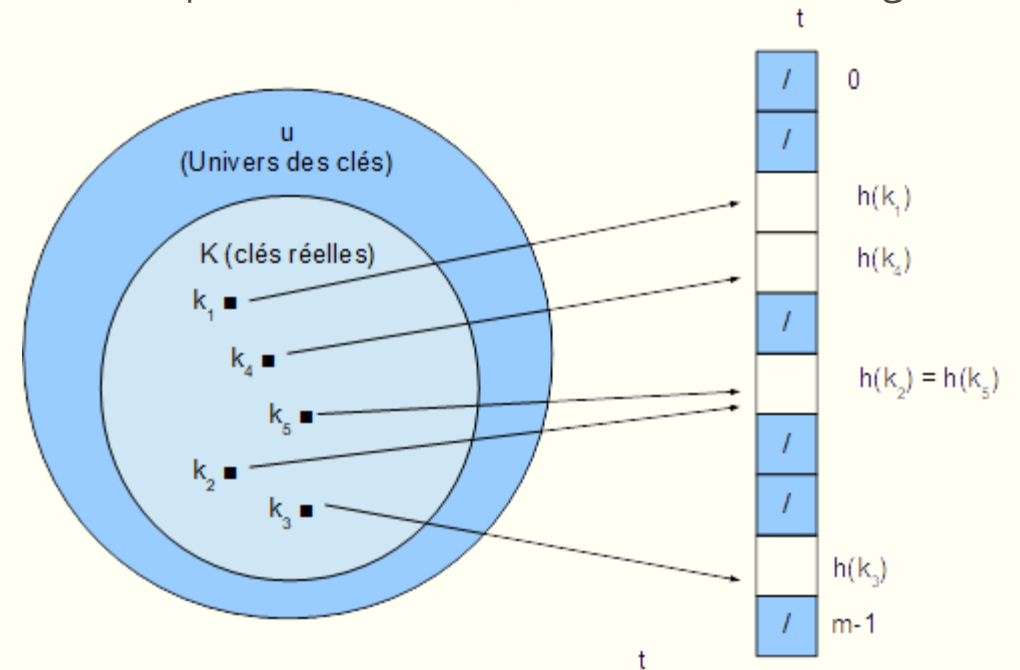
▪ SUPPRIMER-ADRESSAGE-DIRECT (t, x)

Limites de l'adressage direct

- Si l'univers \mathcal{U} est trop grand
 - Gérer une table de taille $|\mathcal{U}|$???
 - Taille mémoire !!!
- L'ensemble k des clés réellement stockées peut être petit comparé à \mathcal{U}
 - Gaspillage de l'espace mémoire
- Solution : table de hachage

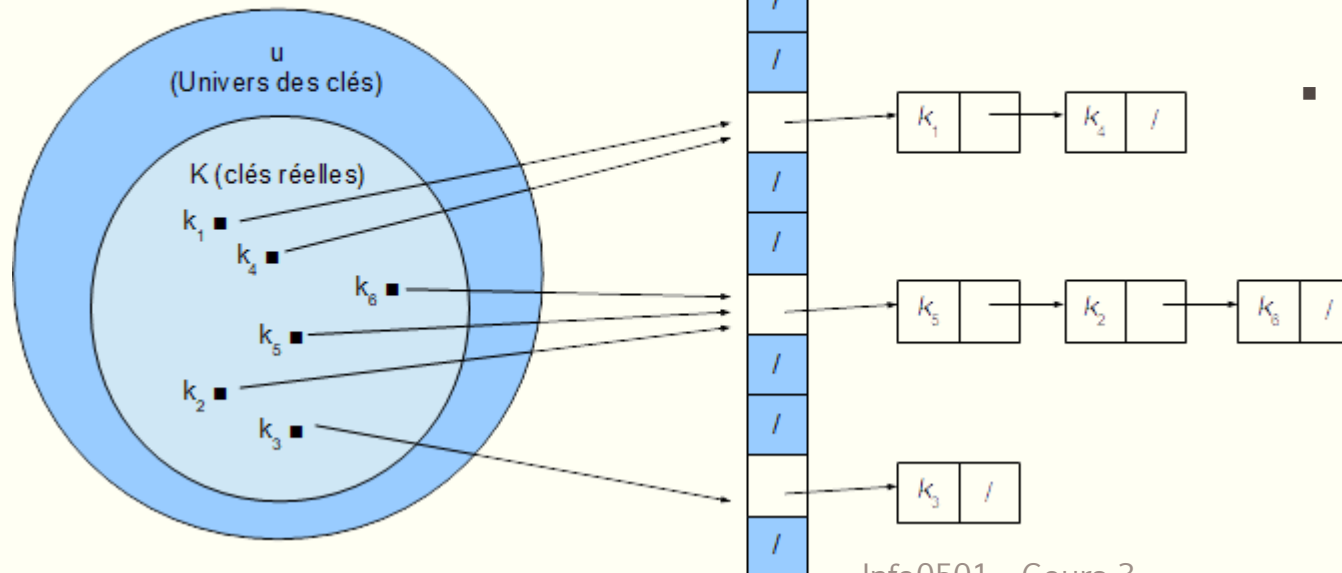
Table de hachage

- Lorsque l'ensemble k des clés stockées est petit comparé à u
 - Requier moins d'espace de stockage qu'une table à adressage direct
 - ... tout en conservant la recherche en $O(1)$ (moyen)
- Éléments de clé k
 - Stocké dans l'alvéole $h(k)$
- On utilise une **fonction de hachage** h pour calculer l'alvéole à partir de la clé k
- h établit une correspondance entre l'univers u des clés et les alvéoles d'une table de hachage $t[0 .. m - 1]$
 - m : taille de la table de hachage
 - Un élément de clé k est *haché* dans l'alvéole $h(k)$
- Plusieurs clés peuvent être hachées vers la même alvéole
 - Il y a alors **collision**
- Pour résoudre les collisions
 - Techniques de résolution, fonctions de hachage



Résolution des collisions par chaînage

- Éléments hachés vers la même alvéole
 - Insérés dans une même liste chaînée
- L'alvéole j
 - Contient un pointeur vers la tête de liste de tous les éléments hachés vers j
 - ... ou *NIL*



Exemple 1

- INSÉRER-HACHAGE-CHAINÉE (t, x)
[Redacted]
- RECHERCHER-HACHAGE-CHAINÉE (t, k)
[Redacted]
- SUPPRIMER-HACHAGE-CHAINÉE (t, x)
[Redacted]

Recherche dans une table de hachage avec chaînage

- Table de hachage t
 - ... à m alvéoles
 - ... qui stocke n éléments
- Facteur de remplissage
 - $\alpha = n / m$ (nb moyen d'éléments d'une chaîne)
- Cas le plus défavorable
 - Les n clés sont hachées dans la même alvéole
 - $\mathcal{O}(n)$ + le temps de calcul de la fonction de hachage
- On s'intéresse plutôt au cas moyen

Recherche dans une table de hachage avec chaînage - Cas moyen

- Les performances dépendent de la répartition en moyenne de l'ensemble des clés à stocker
- On suppose que chaque élément a la même chance d'être haché vers une quelconque alvéole
- Pour $j = 0, 1, \dots, m - 1$
 - Longueur de la liste $t[j] \rightarrow n_j$
 - $n = n_0 + n_1 + \dots + n_{m-1} \rightarrow$ valeur moyenne de n_j
: $E[n_j] = \alpha = n / m$
- Temps requis pour la recherche d'un élément de clé k
 - Dépend linéairement de la longueur de la liste $t[h(k)]$
 - (si la valeur de hachage $h(k)$ peut être calculée en $O(1)$)
- Donc, temps moyen de la recherche
 - Temps de recherche jusqu'à la fin de la liste $t[h(k)]$
 - ... qui a une longueur moyenne $E[n_{h(k)}] = \alpha$
 - = Temps de calcul de $h(k)$ + temps de recherche
 - = $O(1) + O(\alpha) = O(1 + \alpha)$
- Si le nombre d'alvéoles de la table de hachage est au moins proportionnel au nombre d'éléments de la table
 - $n = O(m)$
 - $\alpha = n / m = O(m) / m = O(1)$
- L'efficacité dépend de la fonction de hachage

Fonctions de hachage

- Une bonne fonction de hachage
 - Vérifie l'hypothèse du hachage uniforme simple
 - Chaque clé a autant de chances d'être hachée vers l'une quelconque des m alvéoles, indépendamment des endroits où sont allées les autres clés
 - Minimise le risque que des valeurs « proches » se retrouvent hachées dans la même alvéole
- 3 fonctions de hachage (il en existe d'autres)
 - Méthode de la division
 - Méthode de la multiplication
 - Hachage universel

Méthode de la division

- On fait correspondre une clé k à l'une des m alvéoles en prenant le reste de la division de k par m
 - $h(k) = k \bmod m$
- Très rapide
 - Une opération de division seulement est requise
- Attention au choix de la valeur de m
 - Ex. 2 : $k = \{1,7,15,21,27,29,31,35,43,45,59,67\}$ et $m = 8/9$
- De façon générale
 - Bonne idée \rightarrow nombre premier pas trop proche d'une puissance de 2

Méthode de la multiplication

- 2 étapes
 - On multiplie la clé k par une constante A de l'intervalle $0 < A < 1$ et on extrait la partie décimale
 - On multiplie la valeur obtenue par m et on prend la partie entière du résultat
- $h(k) = \lfloor m (k A \bmod 1) \rfloor$
- Choix de la valeur de m non critique
- Certaines valeurs de A fonctionnent mieux
 - Une bonne valeur $\rightarrow (\sqrt{5} - 1) / 2 = 0,6180339887...$

Hachage universel

- Si un ennemi choisit les clés à hacher par une fonction de hachage spécifique
 - Il peut choisir n clés qui seront toutes hachées vers la même alvéole
 - Recherche $\rightarrow O(n)$ ☹
- Toute fonction de hachage fixée à l'avance est vulnérable à ce type de comportement
 - Cas le plus défavorable
 - Une bonne idée est alors de choisir la fonction de hachage aléatoirement parmi plusieurs

Adressage ouvert

- Tous les éléments occupent la table de hachage elle-même
- Pour rechercher un élément
 - On examine les alvéoles de la table jusqu'à trouver l'élément ou réaliser qu'elle ne s'y trouve pas
- Aucune information conservée hors de la table
 - Pas de listes chaînées
- Permet d'éviter le recours aux pointeurs
 - On calcule la séquence des alvéoles à examiner

Insertion dans une table de hachage avec adressage ouvert

- On examine successivement (sonde) la table de hachage jusqu'à trouver une alvéole vide
- Au lieu de suivre l'ordre $0, 1, \dots, m - 1$ ($O(n)$)
 - La séquence des positions sondées dépend de la clé
- La séquence de sondage
 - $\langle h(k, 0), h(k, 1), \dots, h(k, m - 1) \rangle$
 - ... doit être une permutation de $\langle 0, 1, \dots, m - 1 \rangle$ (chaque position doit être considérée)
- Différentes techniques de sondage
 - Linéaire, quadratique, double hachage

Sondage linéaire

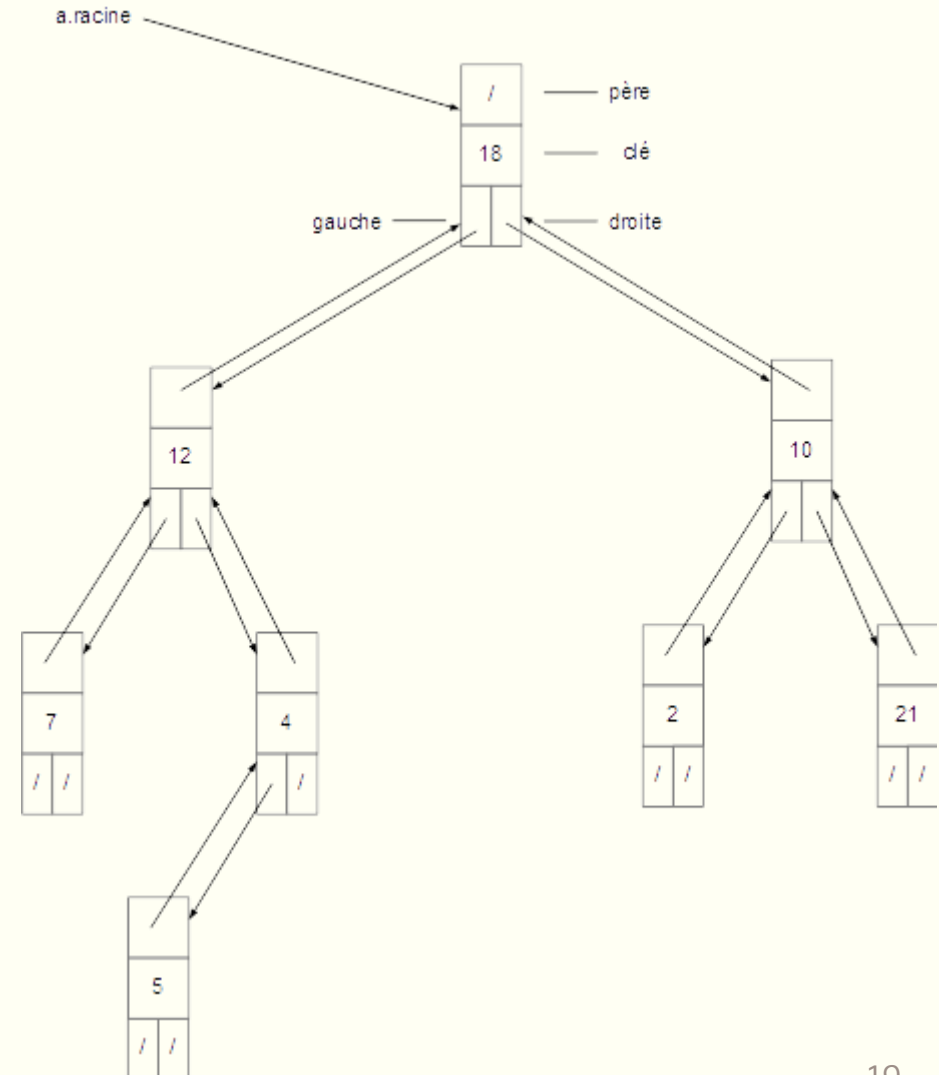
- Étant donnée une fonction de hachage ordinaire h'
 - Appelée fonction de hachage auxiliaire
- ... utilise la fonction de hachage
 - $h(k, i) = (h'(k) + i) \bmod m$
 - pour $i = 0, 1, \dots, m - 1$
- Exemple 3 : insertion des clés
 - $k = \{10, 22, 31, 4, 15, 28, 17, 88, 59\}$
 - $m = 11$ et $h'(k)$: méthode de la division



ARBRES BINAIRES DE RECHERCHE

Arbre binaire de recherche

- Structure de données pouvant supporter plusieurs opérations d'ensemble dynamique
 - RECHERCHER, MINIMUM, MAXIMUM, PRÉDÉCESSEUR, SUCCESSEUR, INSÉRER, SUPPRIMER
 - Temps proportionnel à la hauteur de l'arbre
 - $O(\lg n)$
 - Peut servir aussi bien de dictionnaire que de file de priorités
- Représentation chaînée
- Chaque nœud est un objet
- Attributs de chaque noeud
 - clé, données satellites, gauche, droite, pere
- Attribut de l'arbre
 - Racine

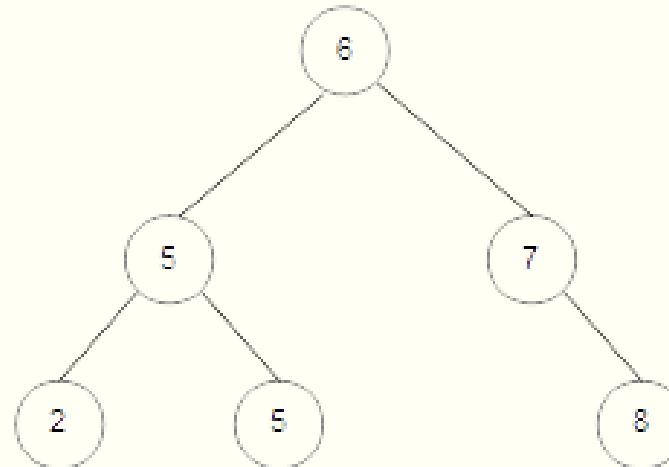


Propriété d'arbre binaire de recherche

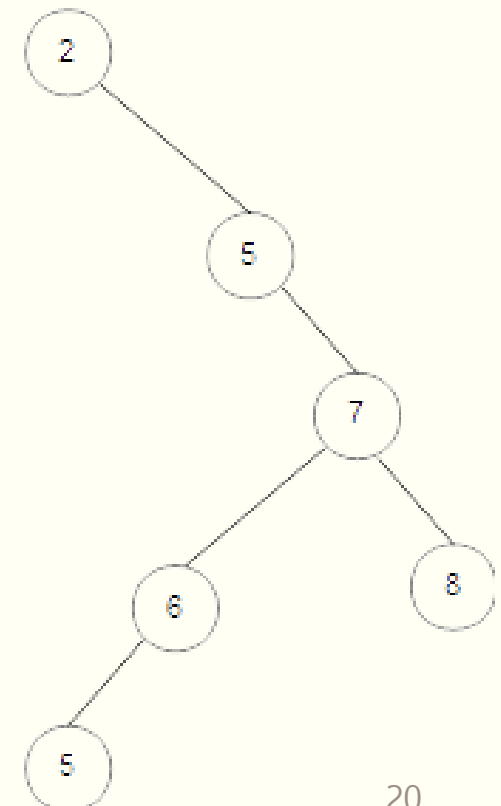
- Les clés d'un arbre binaire de recherche sont toujours stockées de manière à satisfaire la **propriété d'arbre binaire de recherche** :
- Soit x un nœud d'un arbre binaire de recherche
 - Si y est un nœud du sous-arbre de gauche de x
 - Alors $y.clé \leq x.clé$
 - Si y est un nœud du sous-arbre de droite de x
 - Alors $x.clé \leq y.clé$

- Arbre binaire de recherche avec $k = \{2, 5, 5, 6, 7, 8\}$

de hauteur 2



de hauteur 4



Affichage d'un arbre binaire de recherche

- Dépend de l'ordre de parcours des noeuds
- Parcours infixe
 - Affiche la clé de la racine entre les clés du sous-arbre de gauche et les clés du sous-arbre de droite
- Parcours préfixe
 - Affiche la racine avant les sous-arbres
- Parcours postfixe
 - Affiche la racine après les sous-arbres
- Exemple 4

Requêtes dans un arbre binaire de recherche

- Rechercher
 - Récursif, Itératif
- Minimum
- Maximum
- Successeur
- Prédécesseur
- Insertion
- Suppression

Requêtes dans un arbre binaire de recherche

RECHERCHER-ARBRE(x, k)

- On démarre la recherche à la racine
- On suit un chemin simple descendant
 - À chaque nœud, on compare k à $x.clé$
 - Égale \rightarrow Recherche terminée
 - Plus petit \rightarrow on continue dans le sous-arbre de gauche
 - Plus grand \rightarrow on continue dans le sous-arbre de droite
- Exemple 5 – Recherche
- Temps d'exécution
 - $O(h)$ où h est la hauteur de l'arbre

MINIMUM-ARBRE(x)

- On suit les pointeurs *gauche* à partir de la racine jusqu'à ce qu'on rencontre NIL
- MAXIMUM-ARBRE(x)
 - Même chose, mais en suivant les pointeurs *droite*
- Exemple 6-7 : Min et max
- Temps d'exécution
 - $O(h)$

Requêtes dans un arbre binaire de recherche

SUCCESSEUR-ARBRE(x)

- Donne le noeud qui suit x dans un parcours infixe
 - Noeud possédant la plus petite clé supérieure à x
- Pas besoin de comparer les clés
 - Si le sous-arbre de droite n'est pas vide \rightarrow minimum
 - Si le sous-arbre de droite est vide \rightarrow premier ancêtre de x dont l'enfant de gauche est aussi un ancêtre de x
- PRÉDÉCESSEUR-ARBRE(x) \rightarrow Symétrique
- Exemples 8-9 : Successeur et prédécesseur
- Temps d'exécution
 - $O(h)$

Insertion dans un arbre binaire de recherche

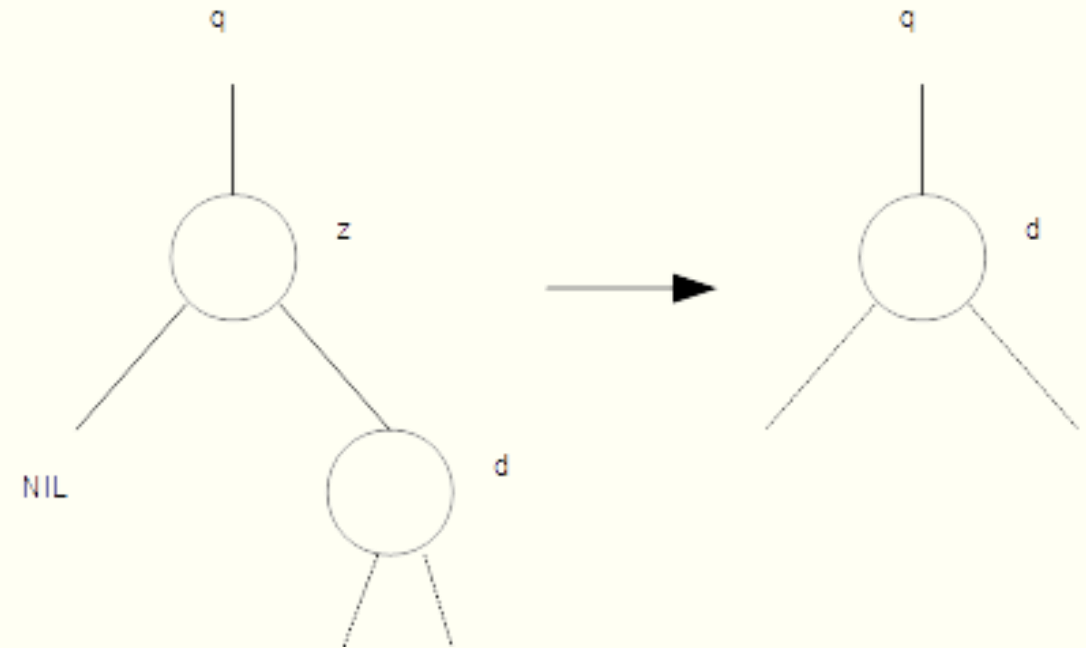
- Modifie la structure de l'arbre
- La propriété d'arbre binaire de recherche doit être conservée après insertion
- Pour insérer une valeur v , on insère un nœud z
 - $z.clé = v$
 - $z.gauche = \text{NIL}$, $z.droite = \text{NIL}$
- On modifie l'arbre et z pour permettre à z d'être inséré à sa position correcte dans l'arbre
- On démarre à la racine de l'arbre
- On insère sous une feuille de l'arbre
 - On descend dans l'arbre
 - ... à droite ou à gauche selon la valeur de $z.clé$
 - ... en cherchant un NIL à remplacer par z
 - La position du NIL trouvé indique où placer z
- Exemple 10 – Insertion
- Temps d'exécution
 - $O(h)$

Suppression dans un arbre binaire de recherche

- On veut supprimer un nœud z de l'arbre
- La propriété d'arbre binaire de recherche doit être conservée après suppression
- On distingue 3 situations possibles
 - z n'a pas d'enfants : on remplace le pointeur vers z de son parent par NIL (facile)
 - z n'a qu'un seul enfant : on remonte l'enfant pour qu'il prenne la place de z dans l'arbre (facile)
 - z a 2 enfants : on trouve le successeur de z et on lui fait prendre la place de z (un peu moins facile)
- Au niveau de l'algorithme, on subdivisera en 4 cas pour gérer ces 3 situations
 - z n'a pas d'enfant gauche
 - z n'a qu'un seul enfant (qui est son enfant gauche)
 - z a 2 enfants et son successeur est son enfant droit
 - z a 2 enfants et son successeur n'est pas son enfant droit

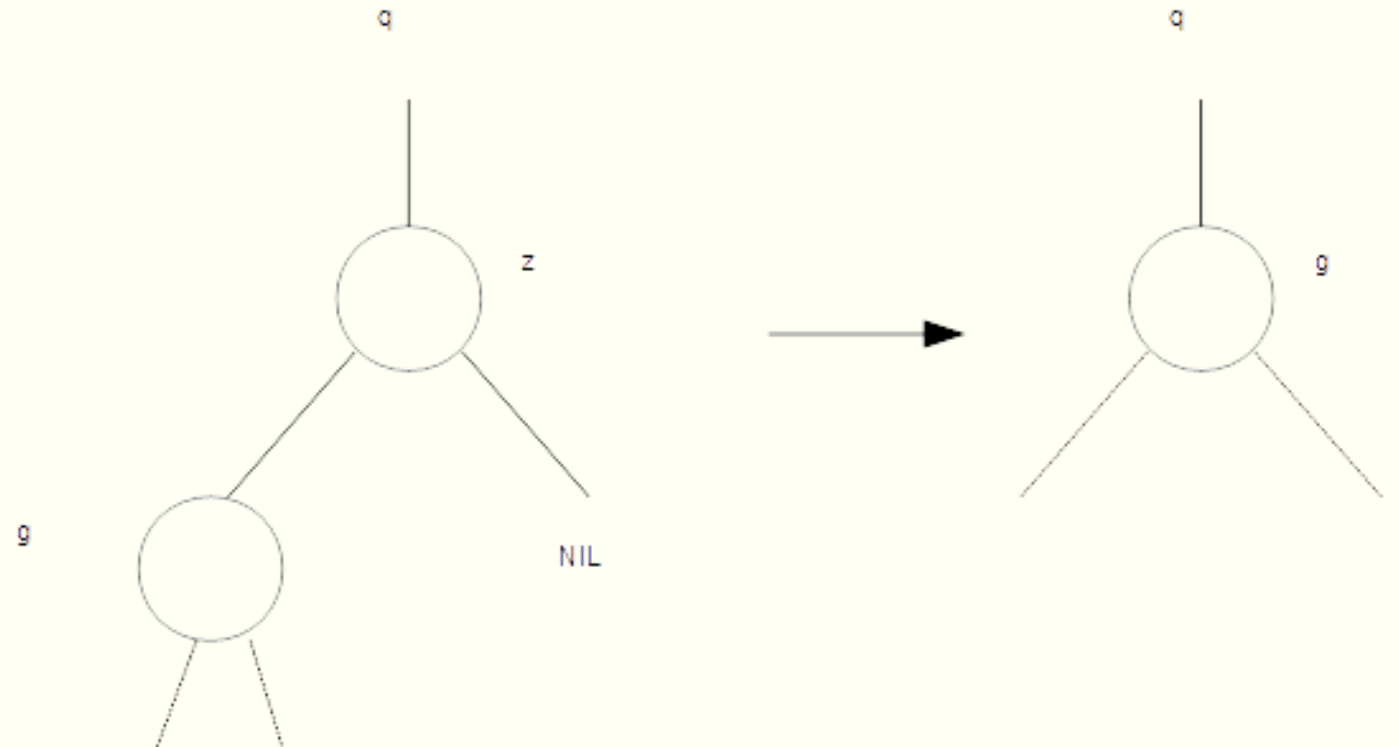
Suppression dans un arbre binaire de recherche

- Cas 1
 - z n'a pas d'enfant gauche
- On remplace z par son enfant droit d
 - Qui est ou n'est pas NIL
 - Si c'est NIL \rightarrow on a géré le cas où z n'a pas d'enfants
- Autrement dit, on transplante le nœud d au nœud z
- Exemple 11
 - Suppression du nœud 15



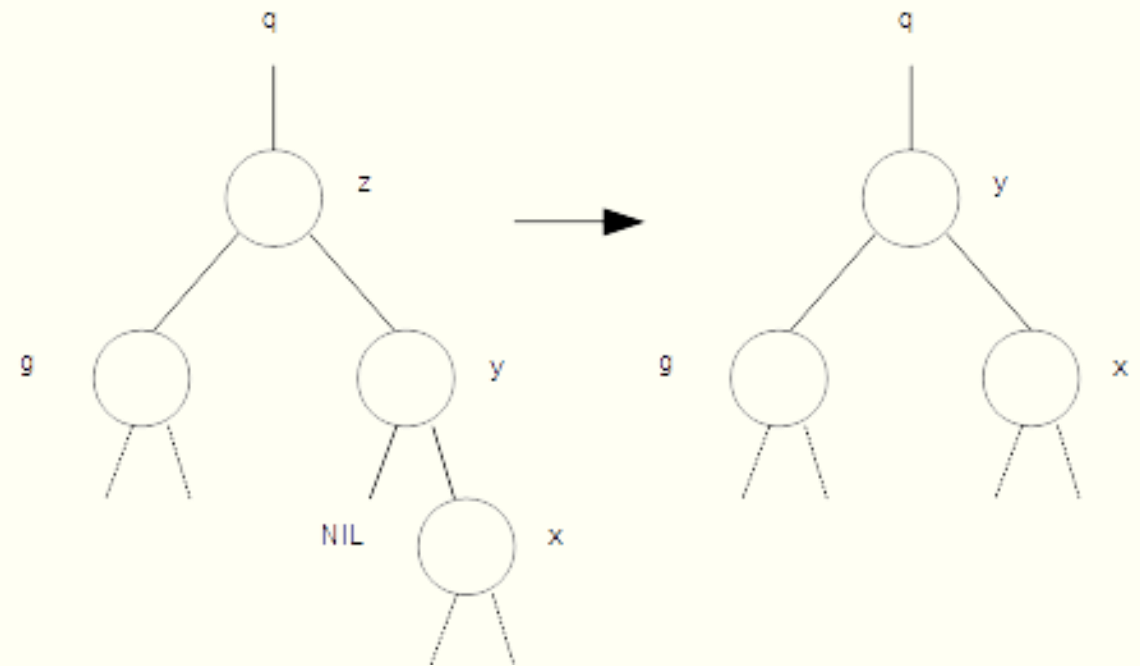
Suppression dans un arbre binaire de recherche

- Cas 2
 - z n'a qu'un seul enfant (qui est son enfant gauche)
- On remplace z par g
- Autrement dit, on transplante le nœud g au nœud z



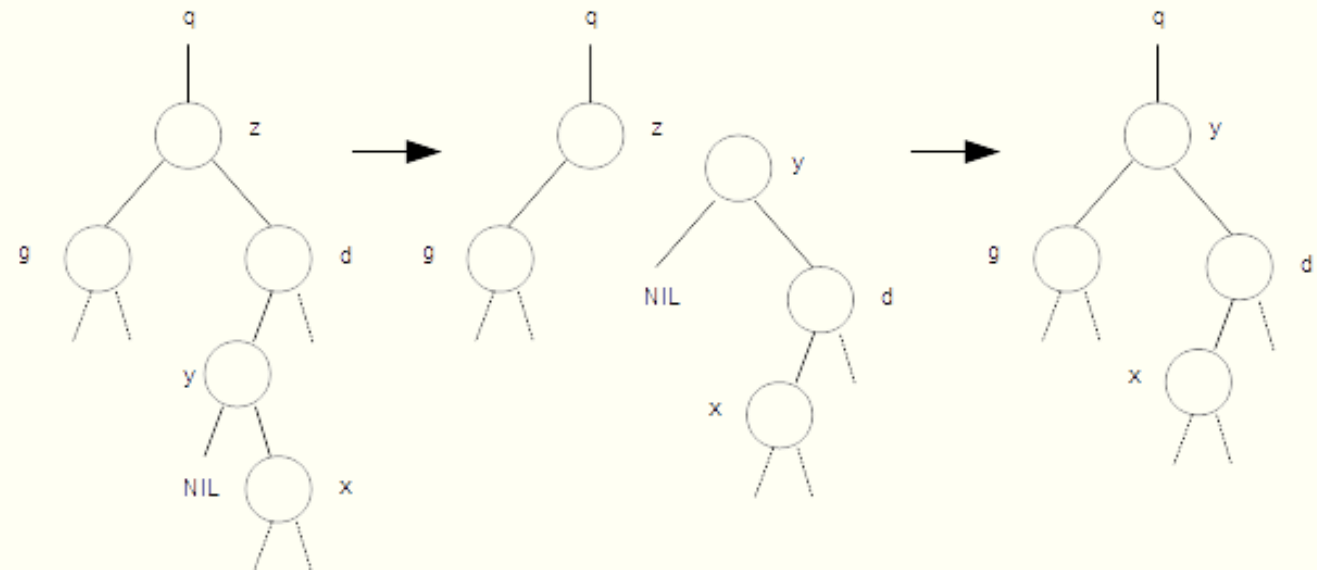
Suppression dans un arbre binaire de recherche

- Cas 3
 - z a 2 enfants et son successeur est son enfant droit
- $g \rightarrow$ enfant gauche de z
- $y \rightarrow$ successeur de z
- $x \rightarrow$ enfant droit de y
- On transplante le nœud y au nœud z
- On modifie le gauche de z pour qu'il soit le gauche de y
- On modifie le père de ce nouveau gauche par y
- Exemple 12
 - Suppression du nœud 5



Suppression dans un arbre binaire de recherche

- Cas 4
 - z a 2 enfants et son successeur n'est pas son enfant droit
- $g \rightarrow$ enfant gauche de z
- $d \rightarrow$ enfant droit de z
- $y \rightarrow$ successeur de z
- $x \rightarrow$ enfant droit de y
- On transplante x au nœud y
- On modifie y pour en faire le parent de d
- On transplante le nœud y au nœud z
- On change le fils gauche de y par g



- On remplace le père de ce fils gauche par y
- Exemple 13
 - Suppression du nœud 12
- Temps d'exécution de la suppression
 - $O(h)$

Synthèse

- Avec un arbre binaire de recherche
 - On peut implémenter l'ensemble des opérations fondamentales d'ensemble dynamique en $O(h)$
- Si la hauteur est petite \rightarrow efficace
- Si la hauteur est grande \rightarrow pas efficace ! (pas mieux qu'une liste chaînée)
- Il est important que l'arbre soit équilibré
 - $O(h) \rightarrow O(\lg n)$
- Une solution \rightarrow en faire un arbre rouge-noir



ARBRES ROUGE-NOIR

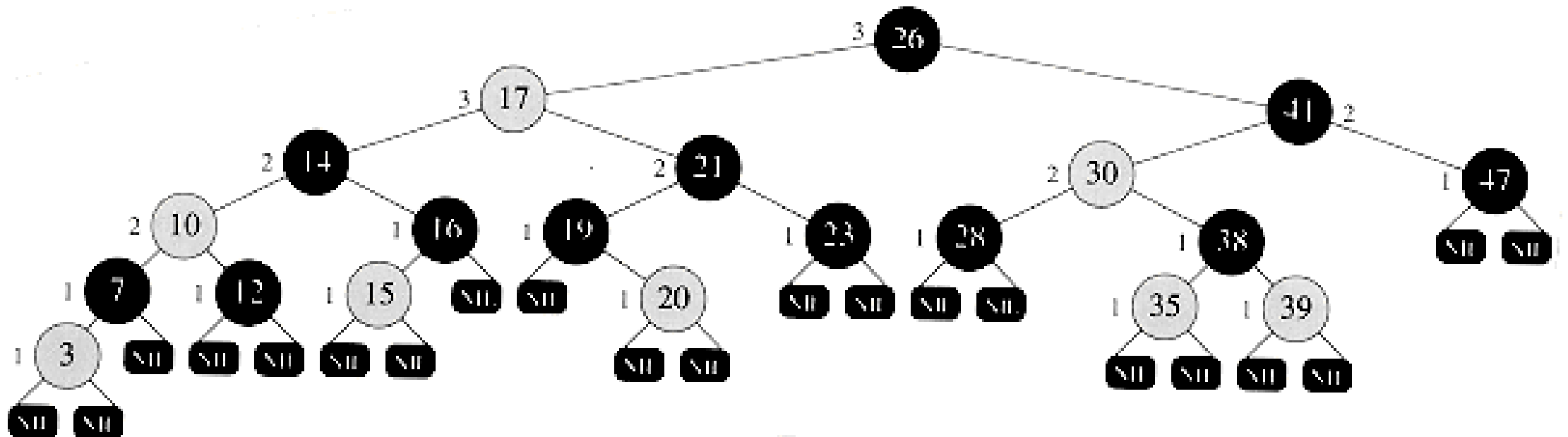
Propriétés des arbres rouge-noir

- Arbre binaire de recherche
- Chaque nœud possède un attribut additionnel
 - Couleur → rouge ou noir
- En contrôlant la manière dont les nœuds sont coloriés sur n'importe quel chemin simple allant de la racine à une feuille
 - On garantit qu'aucun de ces chemins n'est plus de 2 fois plus long que n'importe quel autre
 - Arbre approximativement équilibré

Propriétés des arbres rouge-noir

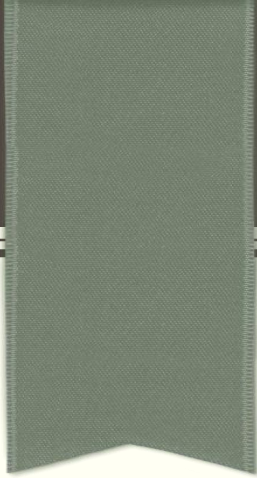
■ Propriétés rouge-noir

- Chaque nœud est soit rouge, soit noir
- La racine est noire
- Chaque feuille (NIL) est noire
- Si un nœud est rouge, alors ses 2 enfants sont noirs
- Pour chaque nœud, tous les chemins simples reliant le nœud à des feuilles situées plus bas contiennent le même nombre de nœuds noirs (hauteur noire)



Propriétés des arbres rouge-noir

- Un arbre rouge-noir ayant n nœuds internes
 - a une hauteur au plus égale à $2 \lg(n + 1)$
- Insertion et suppression
 - Toujours en $\mathcal{O}(\lg n)$
 - Pour conserver les propriétés d'arbre rouge-noir
 - Changement de la couleur de certains nœuds
 - Rotations
- À suivre...



PROCHAIN COURS

GRAPHES