Algorithmique des structures arborescentes

L2 Info et Math-info, 2018–19

Marc Zeitoun

14 février 2019

Moodle

- Nouveaux exercices sur Moodle.
- **Lire** les définitions (polycopié, diapos, **forum**).
- Pour certains exercices, **nombre limité** d'évaluations.

Plan

Plan

Arbres binaires de recherche

Min, Max, Recherche, Insertion, Suppression

Arbres équilibrés

Les arbres rouges et noirs

Les arbres rouges et noirs sont équilibrés

Les arbres binaires de recherche (ABR)

- Partant d'une liste **triée**, on obtient un **arbre particulier**.

Les arbres binaires de recherche (ABR)

- Partant d'une liste **triée**, on obtient un **arbre particulier**.
- En **tout** nœud x :
  - si y est dans le sous-arbre **gauche** de x :

$$\text{étiquette}(y) \leqslant \text{étiquette}(x)$$

  - si y est dans le sous-arbre **droit** de x :

$$\text{étiquette}(y) \geqslant \text{étiquette}(x)$$

Cette propriété ne se vérifie pas uniquement « localement ».

Les arbres binaires de recherche (ABR)

- Partant d'une liste **triée**, on obtient un **arbre particulier**.
- En **tout** nœud x :
  - si y est dans le sous-arbre **gauche** de x :

$$\text{étiquette}(y) \leqslant \text{étiquette}(x)$$

  - si y est dans le sous-arbre **droit** de x :

$$\text{étiquette}(y) \geqslant \text{étiquette}(x)$$

 Cette propriété ne se vérifie pas uniquement « localement ».

Un tel arbre est appelé **arbre binaire de recherche (ABR)**,

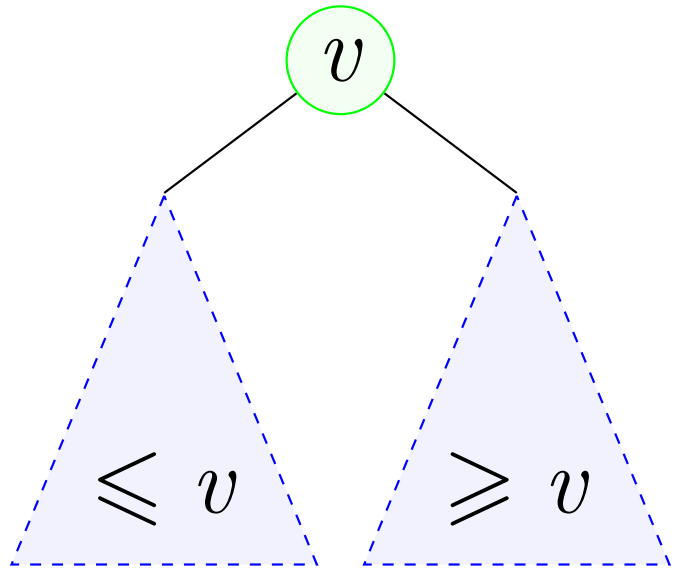
En anglais : **binary search tree (BST)**.

ABR

La propriété d'être un arbre binaire de recherche n'est **pas locale**.

Pour **chaque nœud** v, elle implique :

- **tous les nœuds** du sous-arbre gauche de v,
- **tous les nœuds** du sous-arbre droit de v.

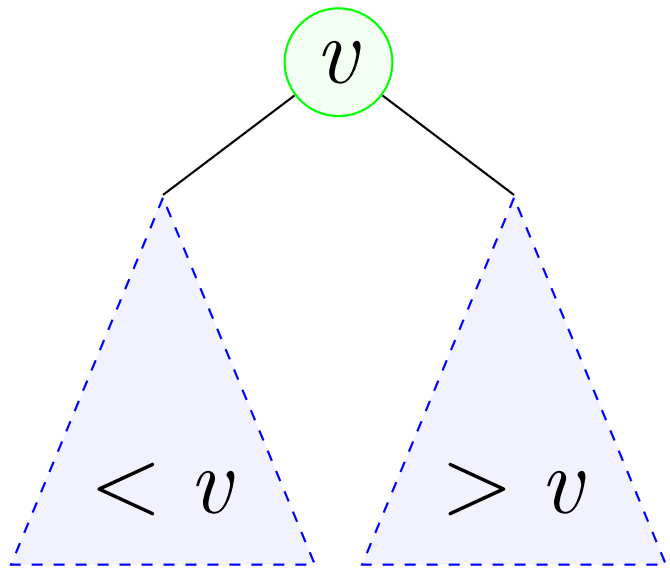


ABR

La propriété d'être un arbre binaire de recherche n'est **pas locale**.

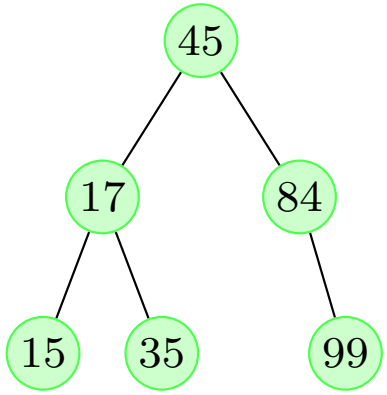
Pour **chaque nœud** v, elle implique :

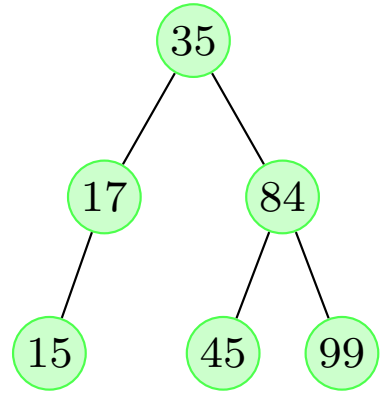
- **tous les nœuds** du sous-arbre gauche de v,
- **tous les nœuds** du sous-arbre droit de v.

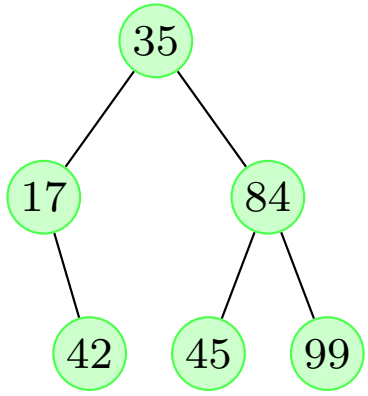


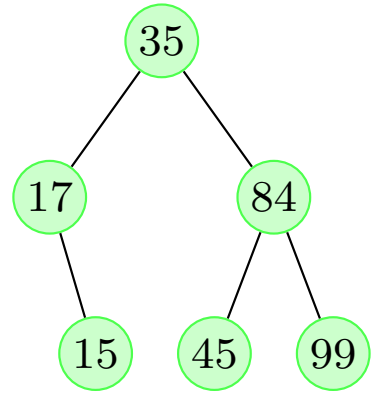
Dans ce cours, 2 nœuds auront toujours des clés différentes

ABR : exemples et non-exemples

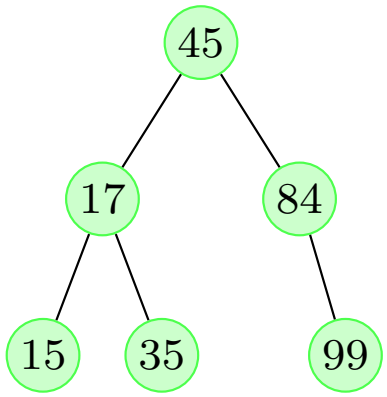




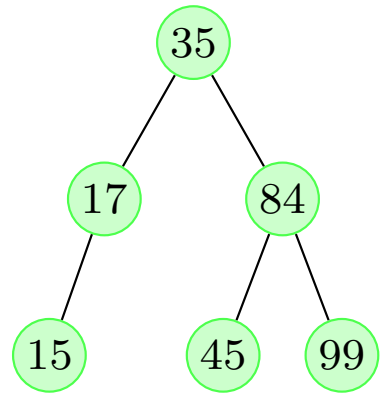


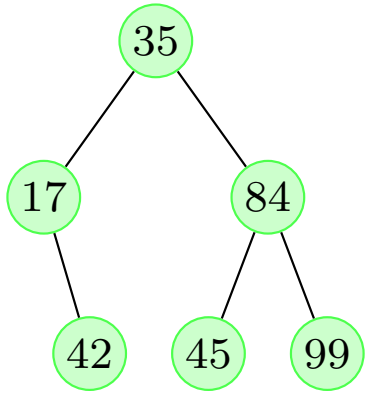


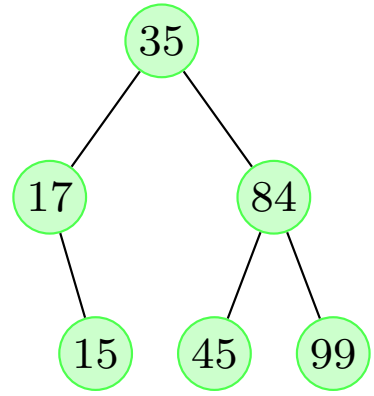
ABR : exemples et non-exemples



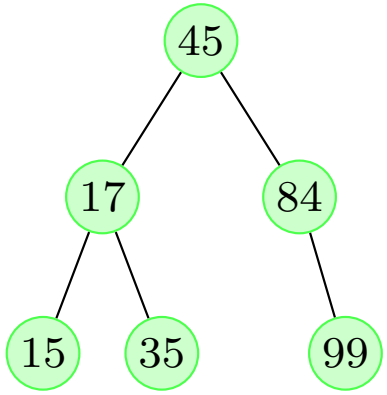
OUI



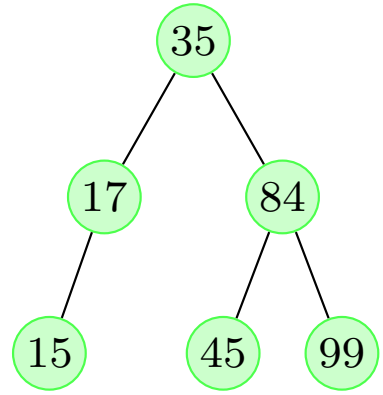




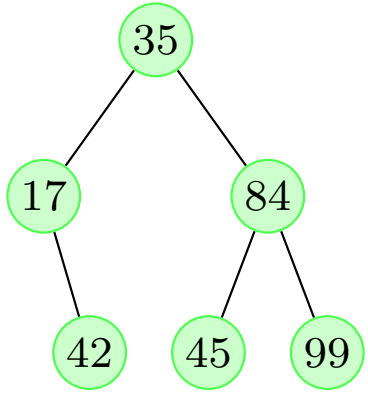
ABR : exemples et non-exemples

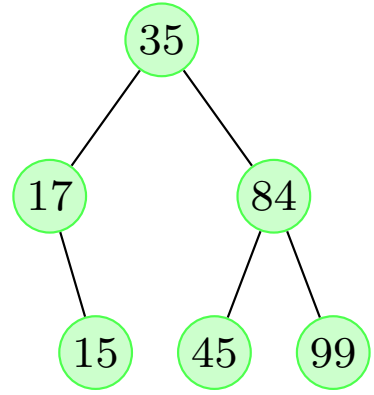


OUI

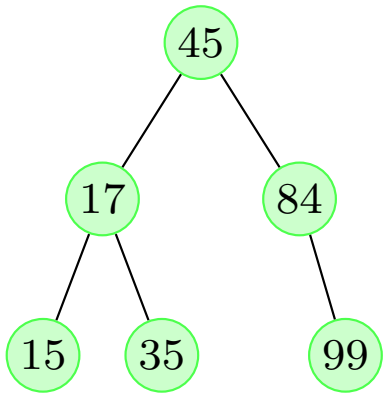


OUI

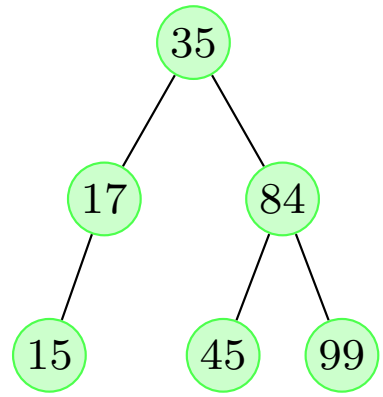




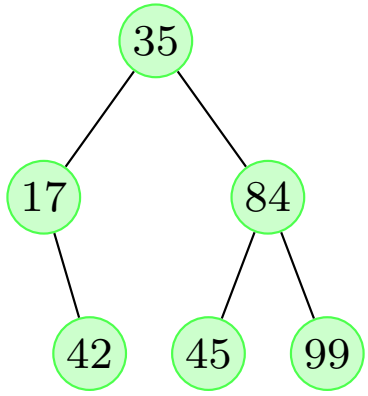
ABR : exemples et non-exemples



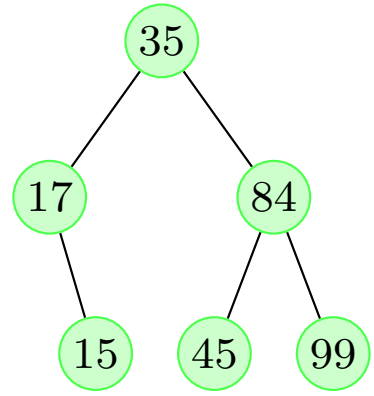
OUI



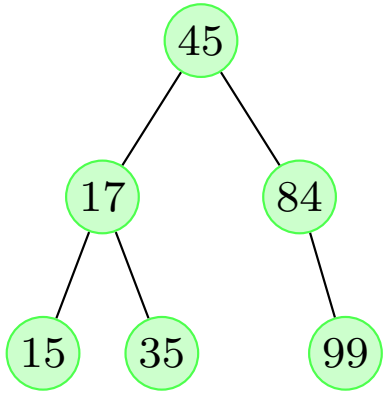
OUI



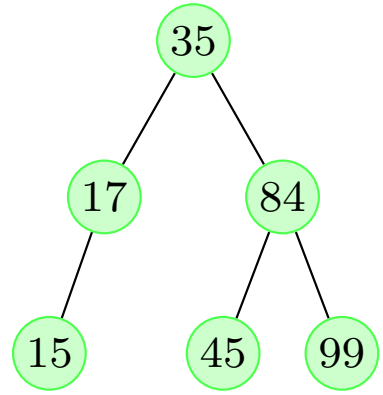
NON



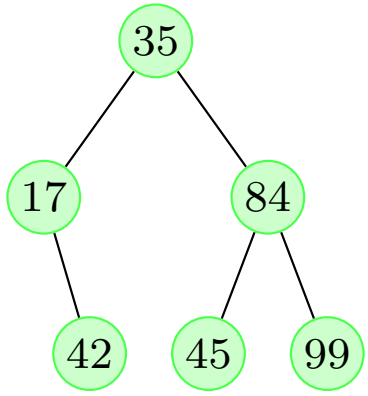
ABR : exemples et non-exemples



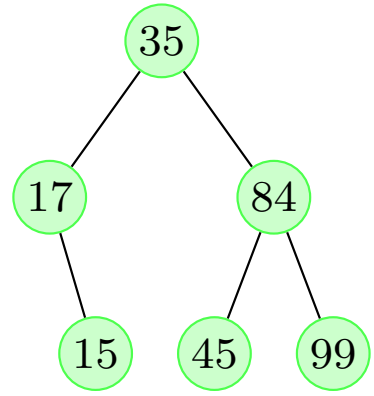
OUI



OUI



NON



NON

Représentation des multi-ensembles par ABR

- Multi-ensemble : fonction f d'un ensemble E dans $\mathbb{N}$.

  Intuition $f(e) = 4$ signifie : e est *présent en* 4 *exemplaires*.
- « Ensembles » pouvant contenir plusieurs fois la même valeur.
- Notation : $\{\!\{1\,,1\,,4\,,5\,,5\,,5\,,7\,,10\,,42\}\!\}$.

Représentation des multi-ensembles par ABR

- Multi-ensemble : fonction f d'un ensemble E dans $\mathbb{N}$.

  Intuition $f(e) = 4$ signifie : e est *présent en* 4 *exemplaires*.
- « Ensembles » pouvant contenir plusieurs fois la même valeur.
- Notation : $\{\!\{1\,,1\,,4\,,5\,,5\,,5\,,7\,,10\,,42\}\!\}$.
- L'ordre n'est pas important :

$$\{\!\{1, 1, 4, 5, 5, 5, 7, 10, 42\}\!\}$$
$$=$$
$$\{\!\{5, 5, 1, 10, 5, 4, 7, 42, 1\}\!\}$$

Représentation des multi-ensembles par ABR

- Multi-ensemble : fonction f d'un ensemble E dans $\mathbb{N}$.

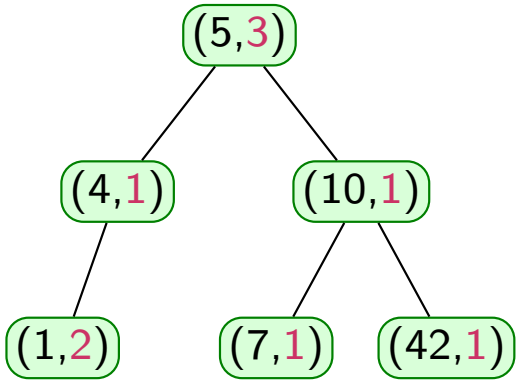
  Intuition $f(e) = 4$ signifie : e est *présent en* 4 *exemplaires*.
- « Ensembles » pouvant contenir plusieurs fois la même valeur.
- Notation : $\{\!\{1\,,1\,,4\,,5\,,5\,,5\,,7\,,10\,,42\}\!\}$.
- L'ordre n'est pas important :

$$\{\!\{1, 1, 4, 5, 5, 5, 7, 10, 42\}\!\}$$
$$=$$
$$\{\!\{5, 5, 1, 10, 5, 4, 7, 42, 1\}\!\}$$

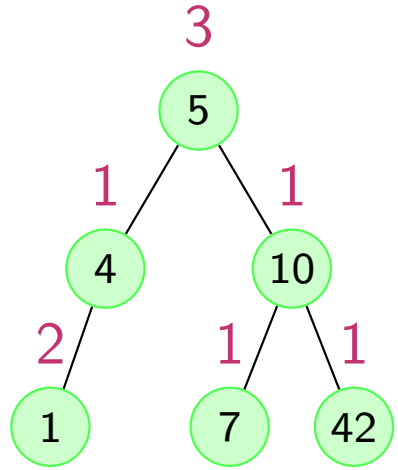
- On utilise les ABR pour représenter les multi-ensembles.
- Élément e en 4 exemplaires $\rightarrow$ nœud d'étiquette $(e\,, 4)$.

  $\Rightarrow$ Aucune clé n'apparaît dans plusieurs nœuds.

Notations



Couples avec clés et multiplicité



Multiplicité au dessus des nœuds

OCaml : `Node((5,3), Node(...), Node(...))`

Objectif

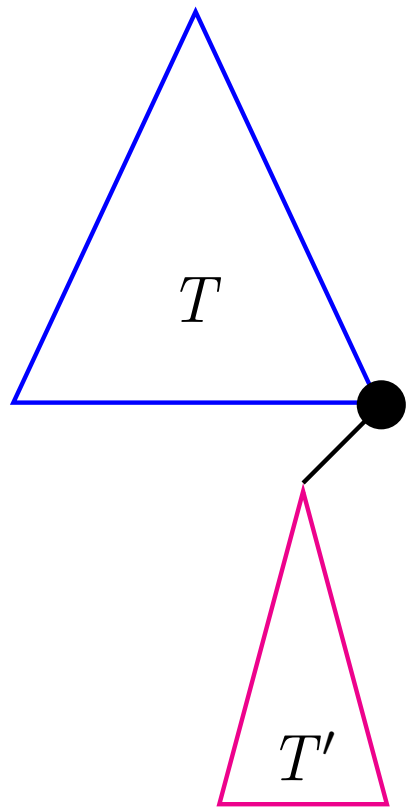
Manipuler des arbres représentant de **grands** multi-ensembles qui évoluent.

On veut pouvoir, rapidement :

- chercher un élément particulier, le min, le max,
- ajouter un élément,
- supprimer un élément.

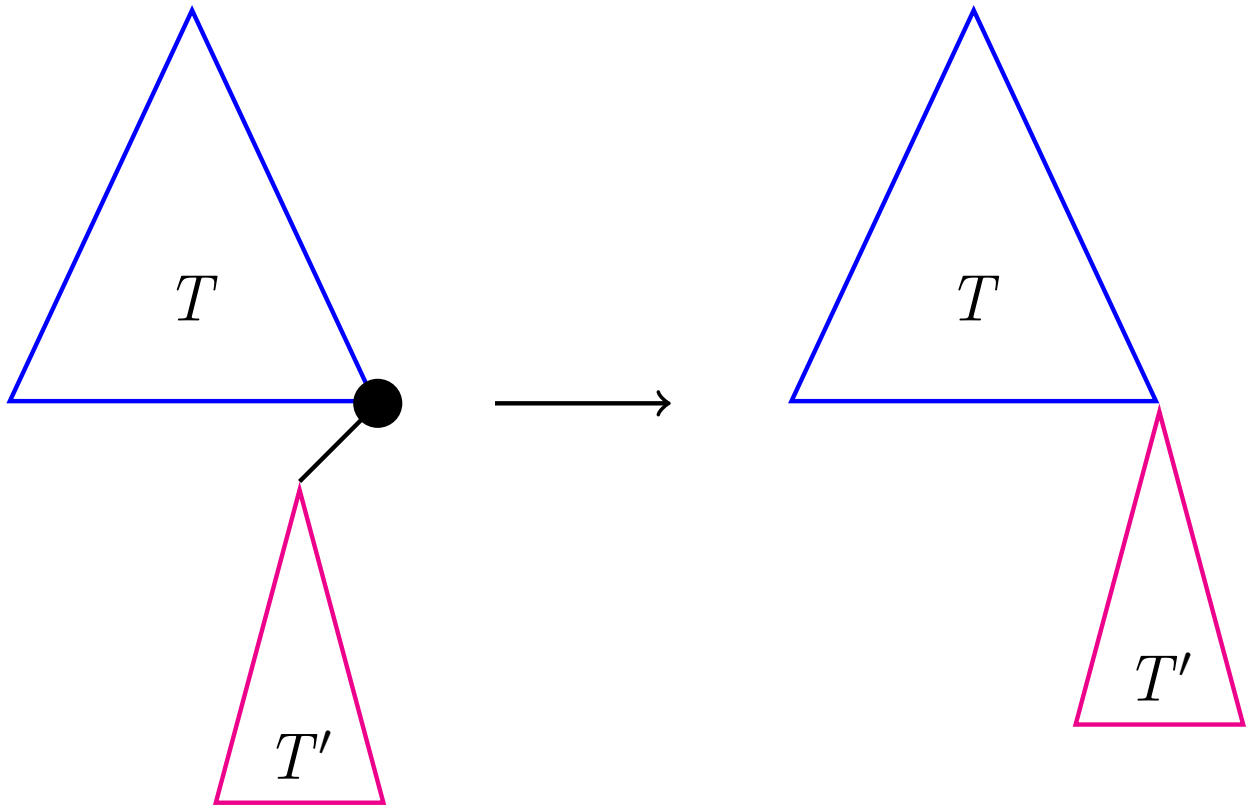
Élément maximum dans un ABR non vide

- Se trouve en parcourant l'arbre depuis la racine « vers la droite ».
- Le nœud portant la clé maximale n'a pas de fils droit.



Suppression de l'élément maximum dans un ABR

- En $O(1)$ une fois le nœud portant la clé maximale localisé.



Recherche d'un élément x

Utilise la propriété d'être un arbre binaire ***de recherche***.

- Si l'arbre est vide : x n'est **pas présent** dans l'arbre.
- Si $x =$ clé(racine) : **trouvé**.
- Si $x <$ clé(racine) : recherche récursive, sous-arbre gauche.
- Si $x >$ clé(racine) : recherche récursive, sous-arbre droit.

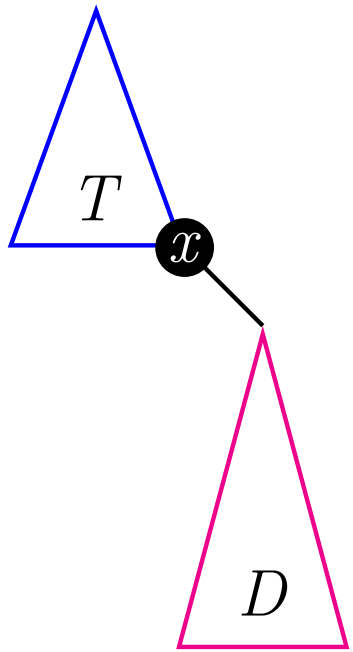
Insertion d'un élément

- Deux algorithmes classiques : par création d'une nouvelle feuille ou par ajout d'une nouvelle racine.
- **Dans ce cours** : par création d'une nouvelle feuille.
- Principe : naviguer jusqu'à
  - soit trouver un nœud portant la clé à insérer (et incrémenter sa multiplicité),
  - soit arriver jusqu'à un sous-arbre vide, et créer une feuille à cet emplacement.

Suppression d'un élément x de multiplicité 1

Principe :

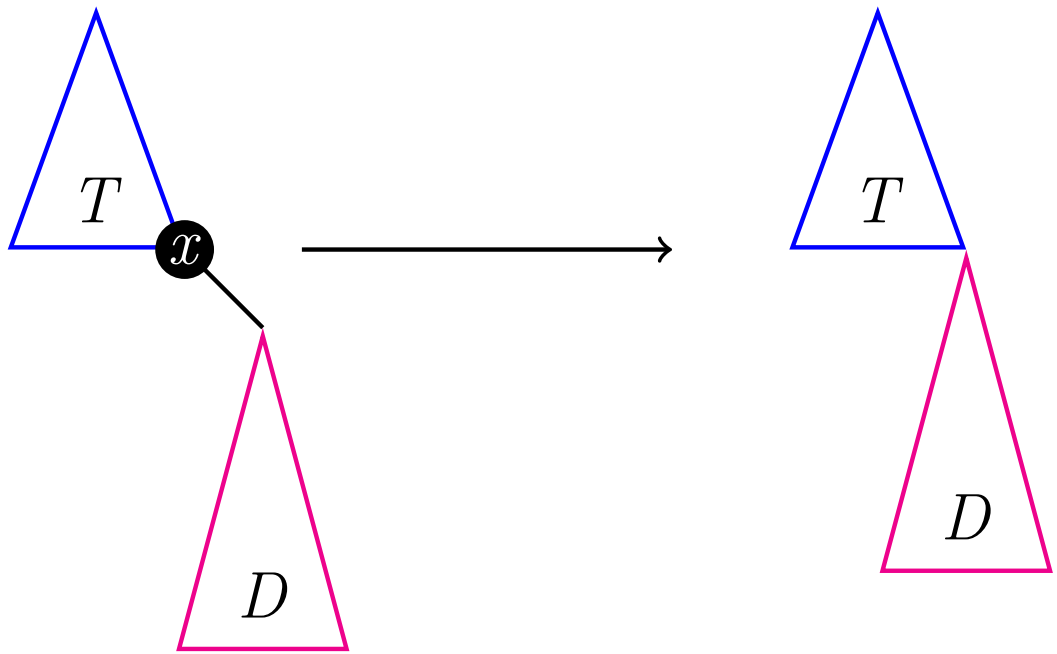
- Rechercher le nœud portant la clé x à supprimer.
- 1er cas : son sous-arbre gauche est vide :



Suppression d'un élément x de multiplicité 1

Principe :

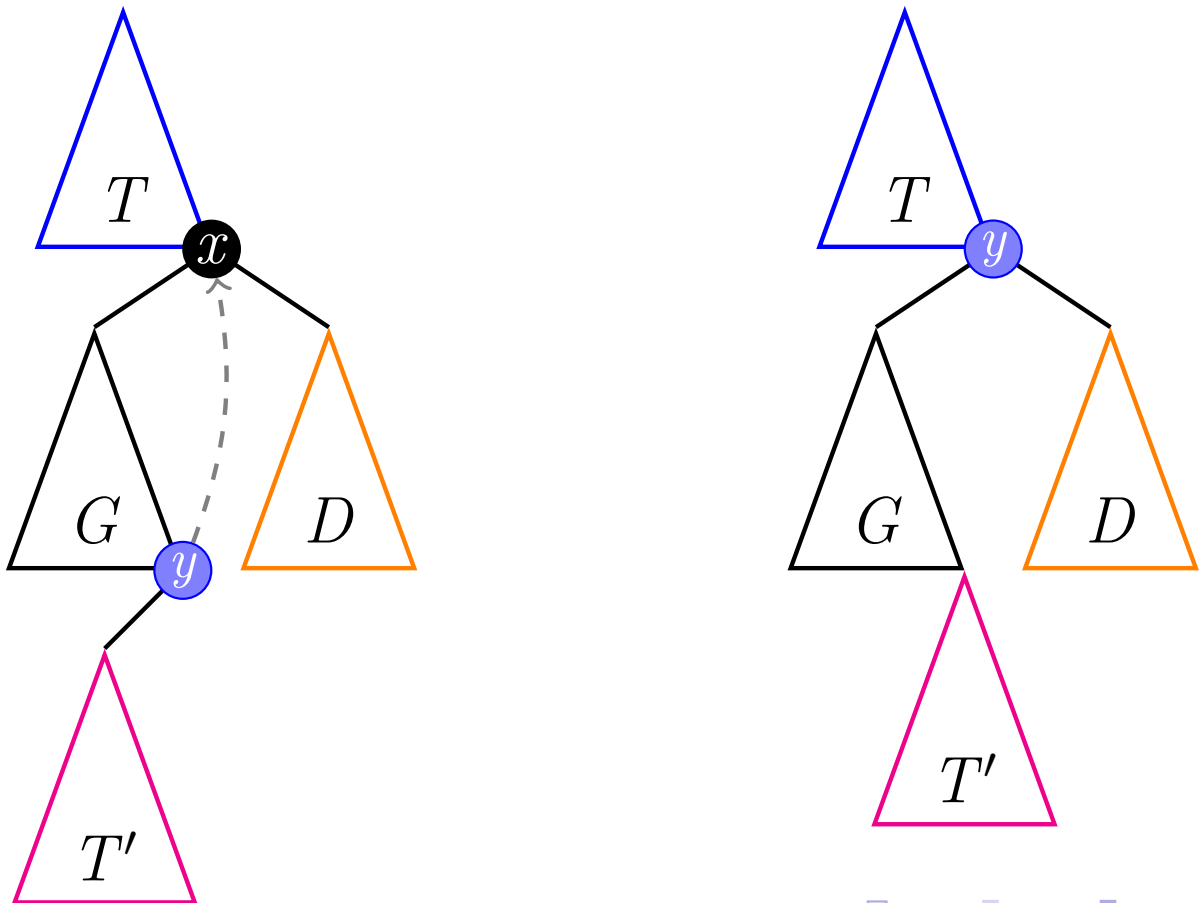
- Rechercher le nœud portant la clé x à supprimer.
- 1er cas : son sous-arbre gauche est vide :



Suppression d'un élément x de multiplicité 1

Principe :

- 2e cas : son sous-arbre gauche est **non** vide : échange avec le maximum y de son sous-arbre gauche, facile à supprimer. **Conserve** la propriété ABR.



Plan

ABR : Complexité recherche/insertion/suppression

- Complexité de recherche/insertion/suppression dans un ABR :

ABR : Complexité recherche/insertion/suppression

- Complexité de recherche/insertion/suppression dans un ABR : $O(h)$ où h est la **hauteur** de l'arbre.

ABR : Complexité recherche/insertion/suppression

- Complexité de recherche/insertion/suppression dans un ABR :

$$O(h) \text{ où } h \text{ est la } \textbf{hauteur} \text{ de l'arbre.}$$

Si n est le nombre de nœuds,

- Au pire, $h =$

ABR : Complexité recherche/insertion/suppression

- Complexité de recherche/insertion/suppression dans un ABR :

$$O(h) \text{ où } h \text{ est la } \textbf{hauteur} \text{ de l'arbre.}$$

Si n est le nombre de nœuds,

- Au pire, $h = n - 1$.
- Au mieux, $h =$

ABR : Complexité recherche/insertion/suppression

- Complexité de recherche/insertion/suppression dans un ABR :

$$O(h) \text{ où } h \text{ est la } \textbf{hauteur} \text{ de l'arbre.}$$

Si n est le nombre de nœuds,

- Au pire, $h = n - 1$.
- Au mieux, $h = \log_2(n+1) - 1$.

Arbres équilibrés

- Maintenir dans les ABR

$$h = \alpha \log(n)$$

garantirait une complexité $O(\log(n))$ **pour ces opérations**.

Arbres équilibrés

- Maintenir dans les ABR

$$h = \alpha \log(n)$$

garantirait une complexité $O(\log(n))$ **pour ces opérations**.

Nouvel objectif

Maintenir $h = \alpha \log(n)$ pour un nombre $\alpha > 0$.

Plan

Les arbres rouges et noirs

Un **arbre rouge et noir** est un ABR tel que :

1. il est **complet**

Les arbres rouges et noirs

Un **arbre rouge et noir** est un ABR tel que :

1. il est **complet** (pas de nœud d'arité 1),

Les arbres rouges et noirs

Un **arbre rouge et noir** est un ABR tel que :

1. il est **complet** (pas de nœud d'arité 1),
2. seuls les nœuds internes portent des valeurs,

Les arbres rouges et noirs

Un **arbre rouge et noir** est un ABR tel que :

1. il est **complet** (pas de nœud d'arité 1),
2. seuls les nœuds internes portent des valeurs,
3. chaque nœud est soit rouge, soit noir,

Les arbres rouges et noirs

Un **arbre rouge et noir** est un ABR tel que :

1. il est **complet** (pas de nœud d'arité 1),
2. seuls les nœuds internes portent des valeurs,
3. chaque nœud est soit rouge, soit noir,
4. la racine est noire,

Les arbres rouges et noirs

Un **arbre rouge et noir** est un ABR tel que :

1. il est **complet** (pas de nœud d'arité 1),
2. seuls les nœuds internes portent des valeurs,
3. chaque nœud est soit rouge, soit noir,
4. la racine est noire,
5. les feuilles sont noires,

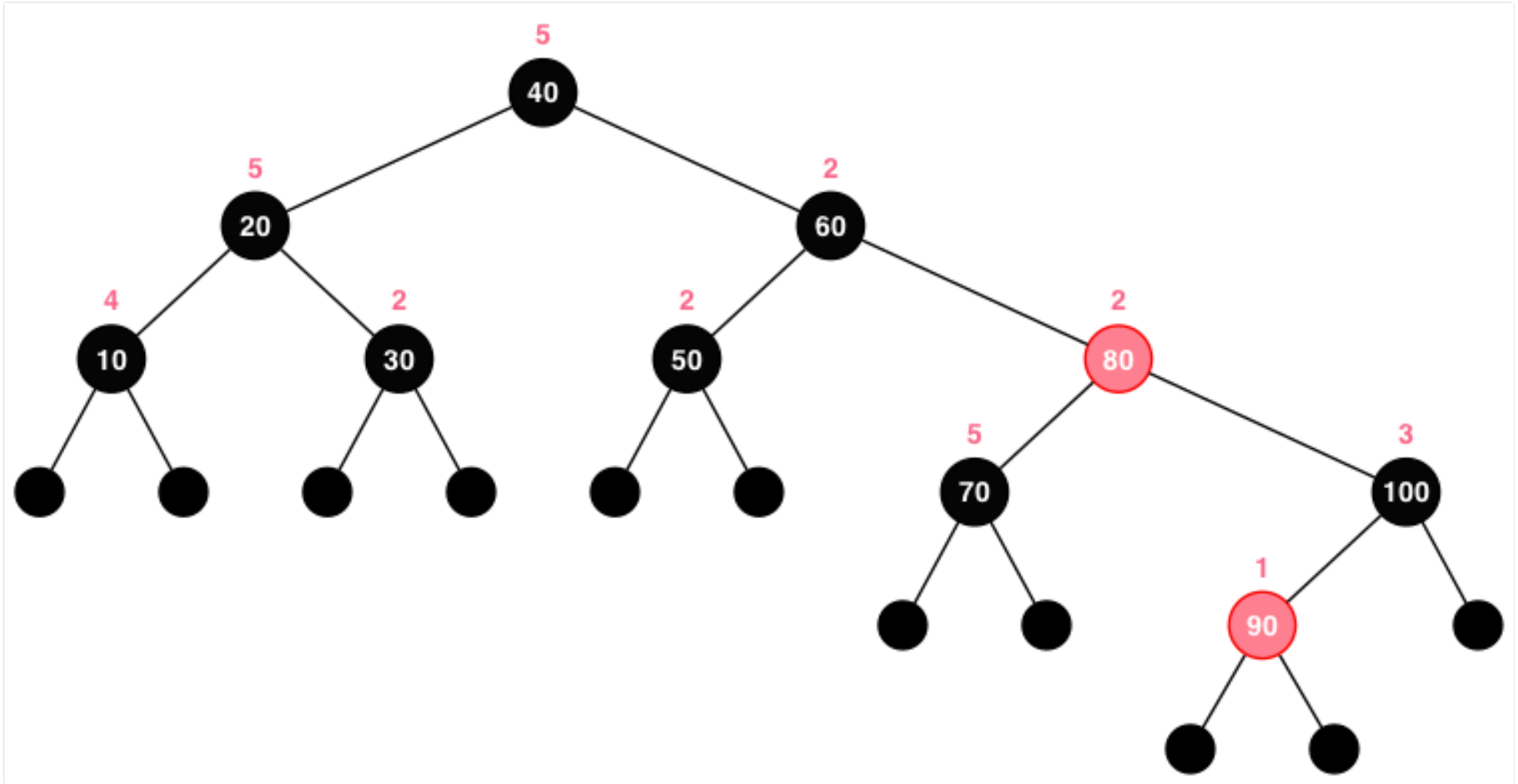
Les arbres rouges et noirs

Un **arbre rouge et noir** est un ABR tel que :

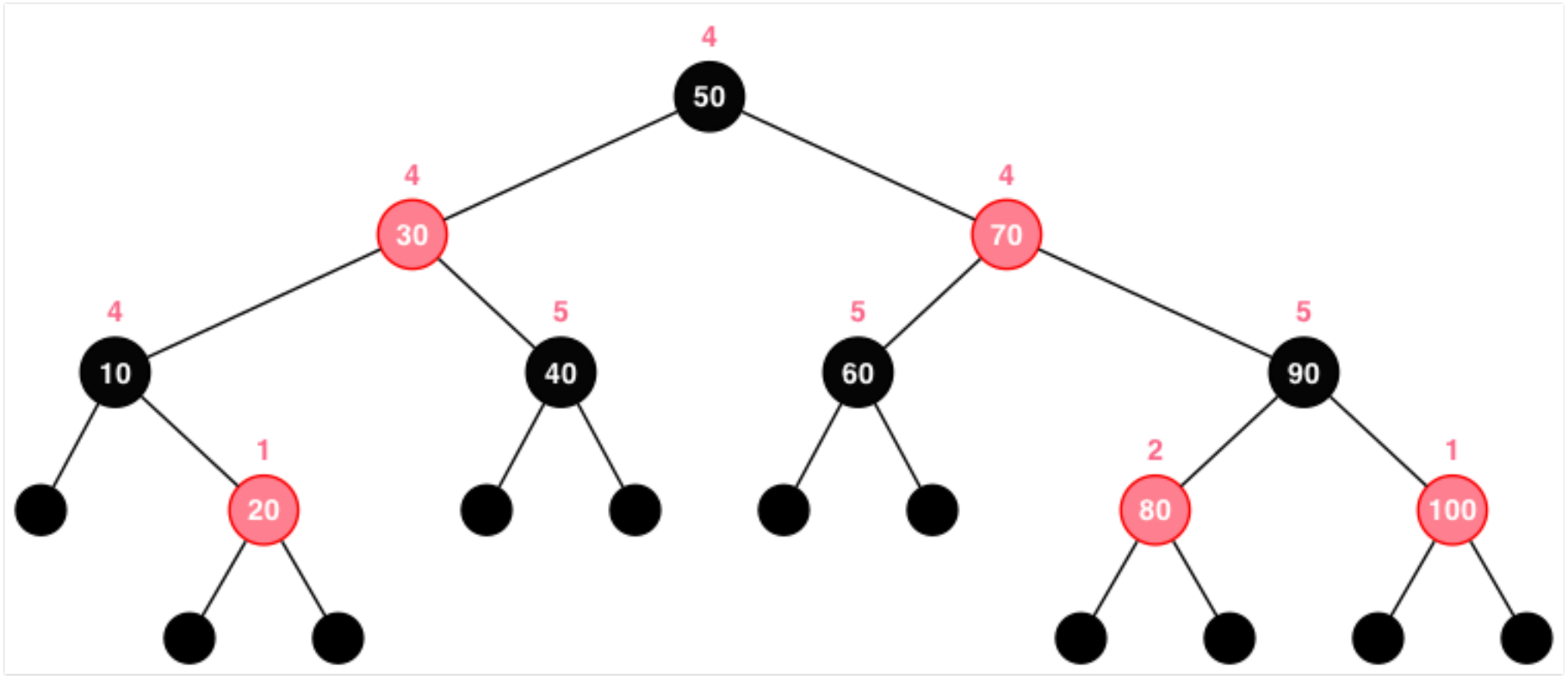
1. il est **complet** (pas de nœud d'arité 1),
2. seuls les nœuds internes portent des valeurs,
3. chaque nœud est soit rouge, soit noir,
4. la racine est noire,
5. les feuilles sont noires,
6. le père d'un nœud rouge est noir.
7. le nombre de nœuds noirs sur chaque branche est **constant**.

Hauteur noire = nombre de nœuds noirs sur chaque branche.

Arbres rouges et noirs : exemple 1



Arbres rouges et noirs : exemple 2



Plan

Les arbres rouges et noirs sont équilibrés

Preuve : sans récurrence. Soit t un arbre rouge et noir.

- $H =$ hauteur noire de t, $h =$ hauteur, $n =$ nombre de nœuds.
- L'arbre est complet et les branches ont au moins H nœuds.

  Donc tous les niveaux sont complets jusqu'à profondeur $H - 1$

Les arbres rouges et noirs sont équilibrés

Preuve : sans récurrence. Soit t un arbre rouge et noir.

- $H =$ hauteur noire de t, $h =$ hauteur, $n =$ nombre de nœuds.
- L'arbre est complet et les branches ont au moins H nœuds.

  Donc tous les niveaux sont complets jusqu'à profondeur $H - 1$
- Donc il y a au minimum $2^H - 1$ nœuds :

  $$2^H - 1 \leqslant n, \text{ donc : } H \leqslant \log_2(n+1).$$

Les arbres rouges et noirs sont équilibrés

Preuve : sans récurrence. Soit t un arbre rouge et noir.

- $H =$ hauteur noire de t, $h =$ hauteur, $n =$ nombre de nœuds.
- L'arbre est complet et les branches ont au moins H nœuds.

  Donc tous les niveaux sont complets jusqu'à profondeur $H - 1$
- Donc il y a au minimum $2^H - 1$ nœuds :

  $$\boldsymbol{2^H - 1 \leqslant n}, \text{ donc : } \boldsymbol{H \leqslant \log_2(n+1).}$$
- Une branche a H nœuds noirs et au maximum $H - 1$ nœuds rouges : $\bullet - \bullet - \bullet - \bullet - \cdots - \bullet - \bullet - \bullet$.
- Donc $h \leqslant 2(H-1)$ et en utilisant l'inégalité ci-dessus :

  $$h \leqslant 2(\log_2(n+1) - 1).$$