Algorithmes sur les arbres binaires de recherche

Benjamin Monmege

Structure de données abstraite : ensemble dynamique d'éléments comparables

- Recherche d'un élément dans un ensemble
- Insertion d'un nouvel élément
- Suppression d'un élément
- Test du vide de l'ensemble

Interface :

```
set.contains(element)     # true or false
set.insert(element)       # modifies set
set.suppress(element)     # modifies set
set.isempty()             # true or false
```

Première implémentation : tableau trié

- Recherche d'un élément dans un ensemble:

Première implémentation : tableau trié

- Recherche d'un élément dans un ensemble:

  recherche dichotomique en $O(\log(n))$

- Insertion d'un nouvel élément:

Première implémentation : tableau trié

- Recherche d'un élément dans un ensemble:

  recherche dichotomique en $O(\log(n))$

- Insertion d'un nouvel élément:

  recherche puis décalage en $O(n)$

- Suppression d'un élément:

Première implémentation : tableau trié

- Recherche d'un élément dans un ensemble:

  recherche dichotomique en $O(\log(n))$

- Insertion d'un nouvel élément:

  recherche puis décalage en $O(n)$

- Suppression d'un élément:

  recherche puis décalage en $O(n)$

- Test du vide de l'ensemble:

Première implémentation : tableau trié

- Recherche d'un élément dans un ensemble:

  recherche dichotomique en $O(\log(n))$

- Insertion d'un nouvel élément:

  recherche puis décalage en $O(n)$

- Suppression d'un élément:

  recherche puis décalage en $O(n)$

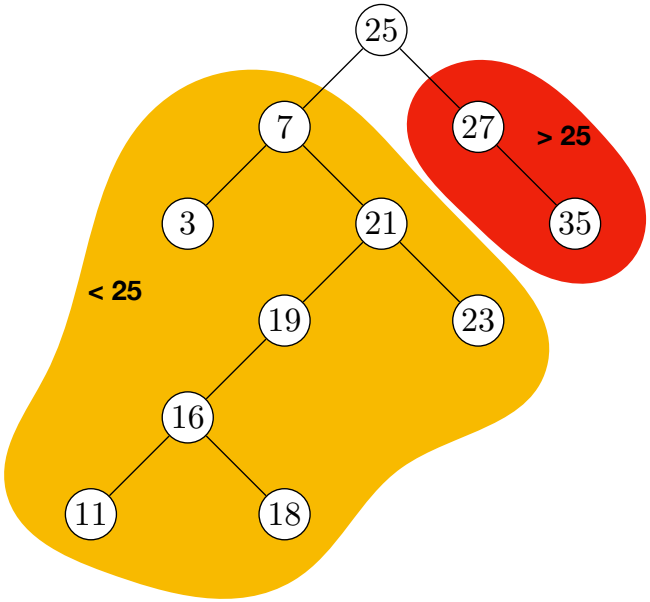
- Test du vide de l'ensemble:

  test de longueur en $O(1)$

Seconde implémentation : Arbre Binaire de Recherche

Arbre binaire étiqueté par les éléments tel que pour tout nœud p d'étiquette x:

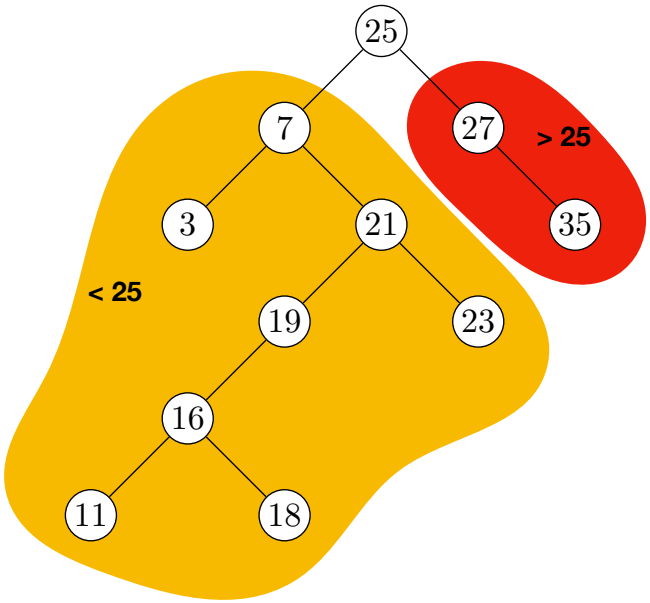
- toutes les étiquettes de l'enfant gauche de p sont inférieures à x
- toutes les étiquettes de l'enfant droit de p sont supérieures à x



Seconde implémentation : Arbre Binaire de Recherche

Arbre binaire étiqueté par les éléments tel que pour tout nœud p d'étiquette x:

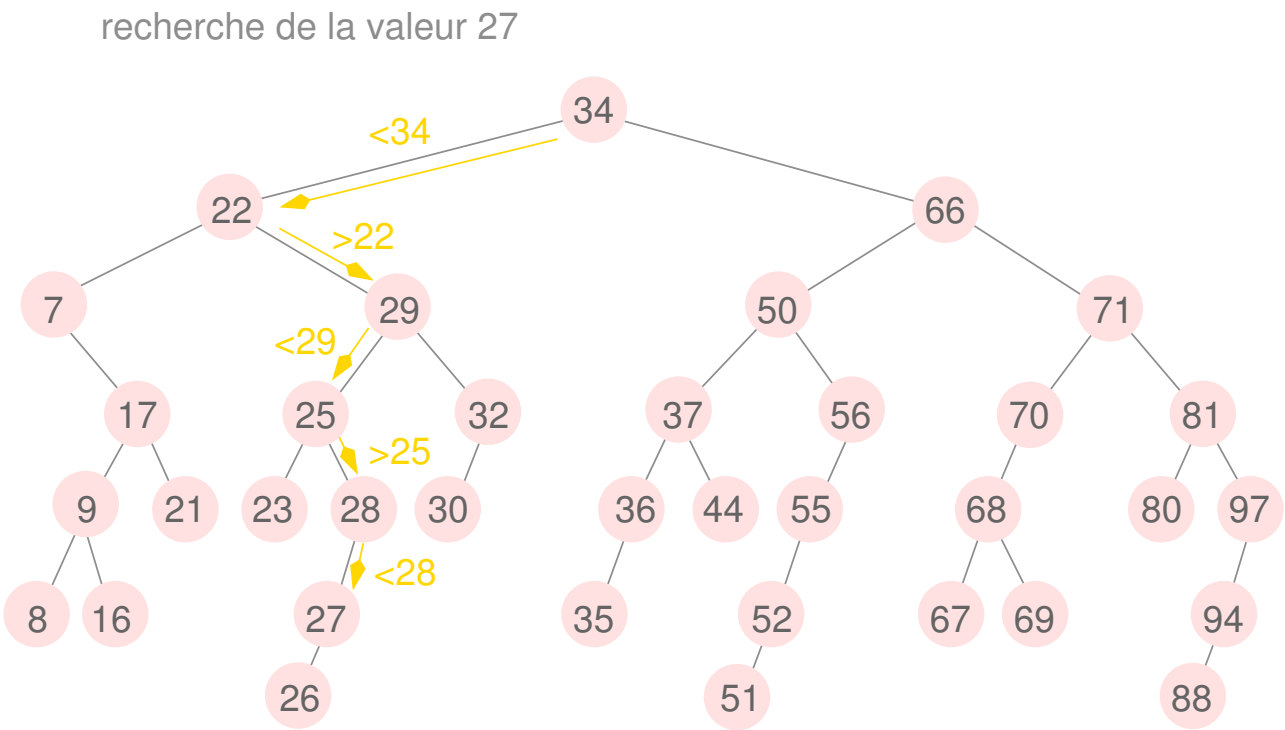
- toutes les étiquettes de l'enfant gauche de p sont inférieures à x
- toutes les étiquettes de l'enfant droit de p sont supérieures à x



Quels algorithmes pour travailler avec ces ABR ?

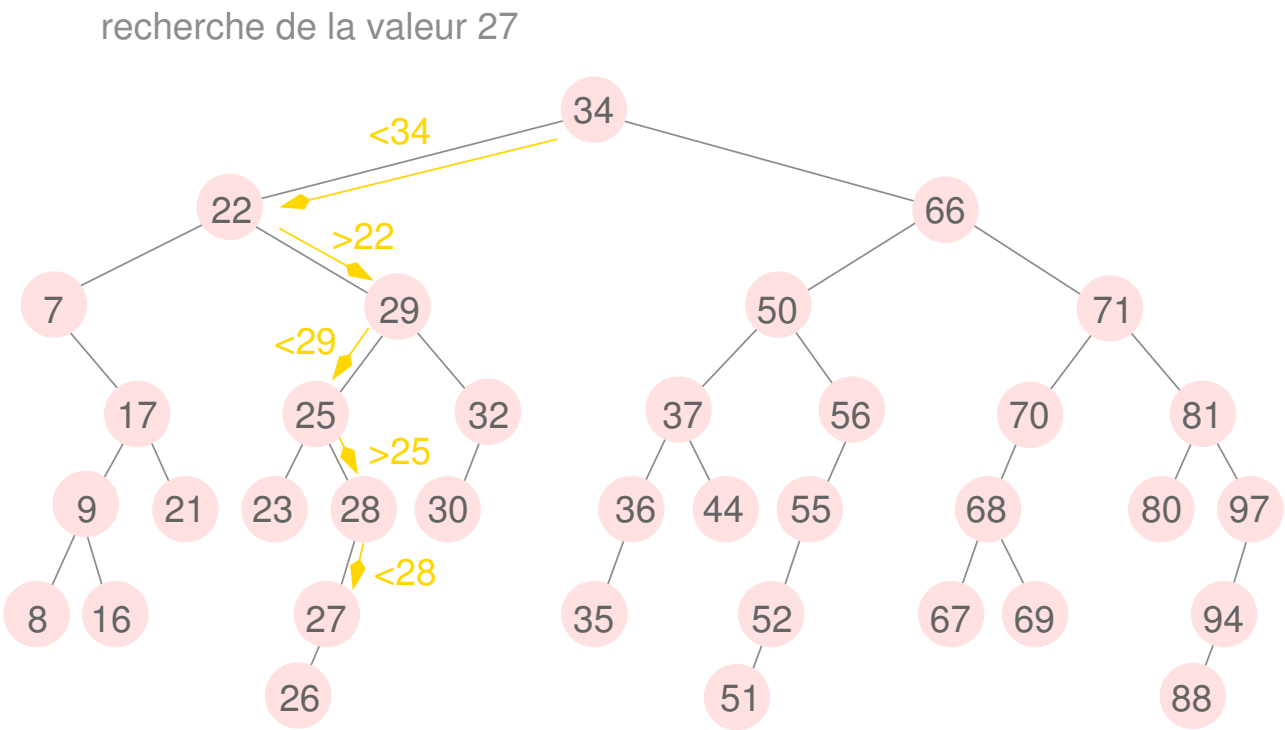
Recherche d'un élément dans un ABR

- Descendre dans l'ABR en suivant l'enfant gauche ou droit



Recherche d'un élément dans un ABR

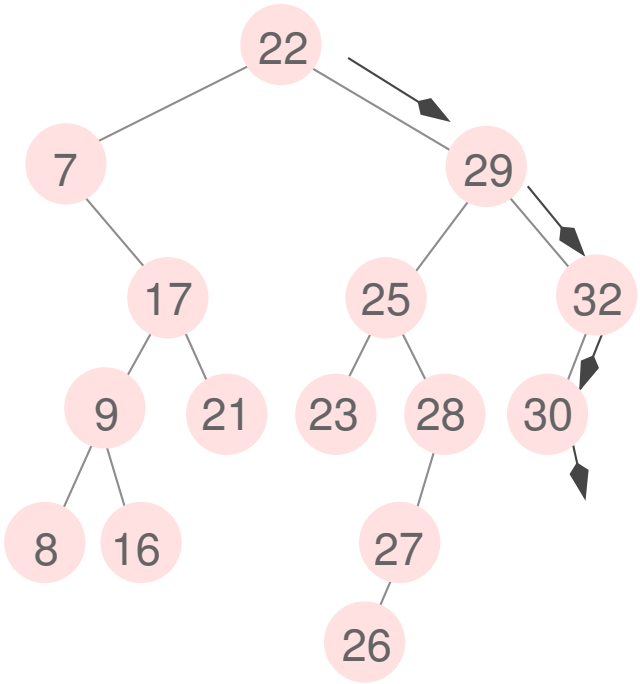
- Descendre dans l'ABR en suivant l'enfant gauche ou droit

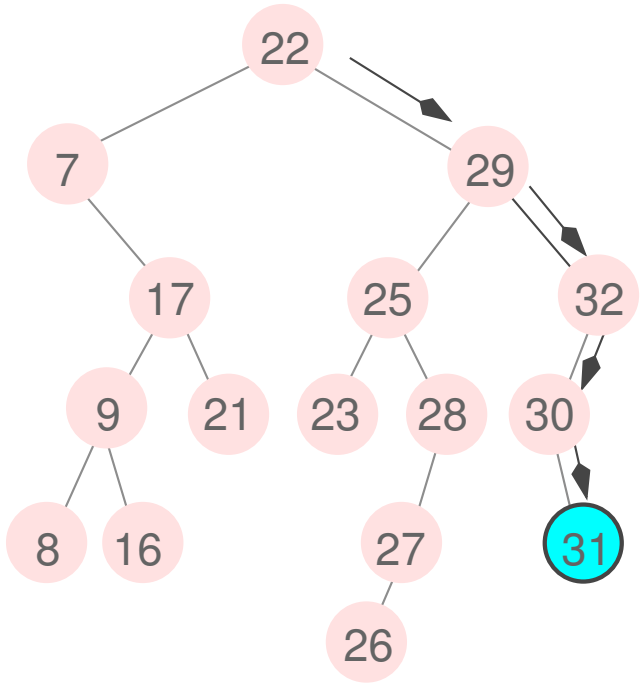


Complexité : $O(hauteur)$

Insertion d'un nouvel élément dans un ABR

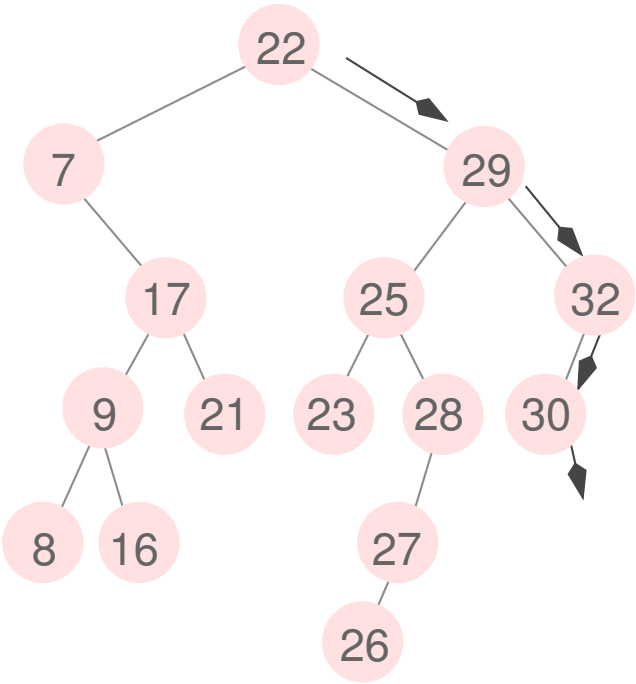
- Recherche puis, en cas d'échec, insertion dans une nouvelle feuille

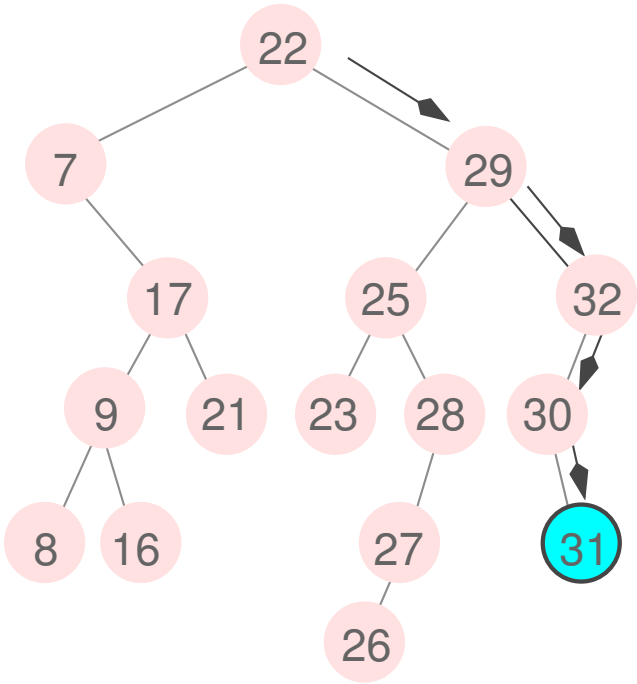




Insertion d'un nouvel élément dans un ABR

- Recherche puis, en cas d'échec, insertion dans une nouvelle feuille

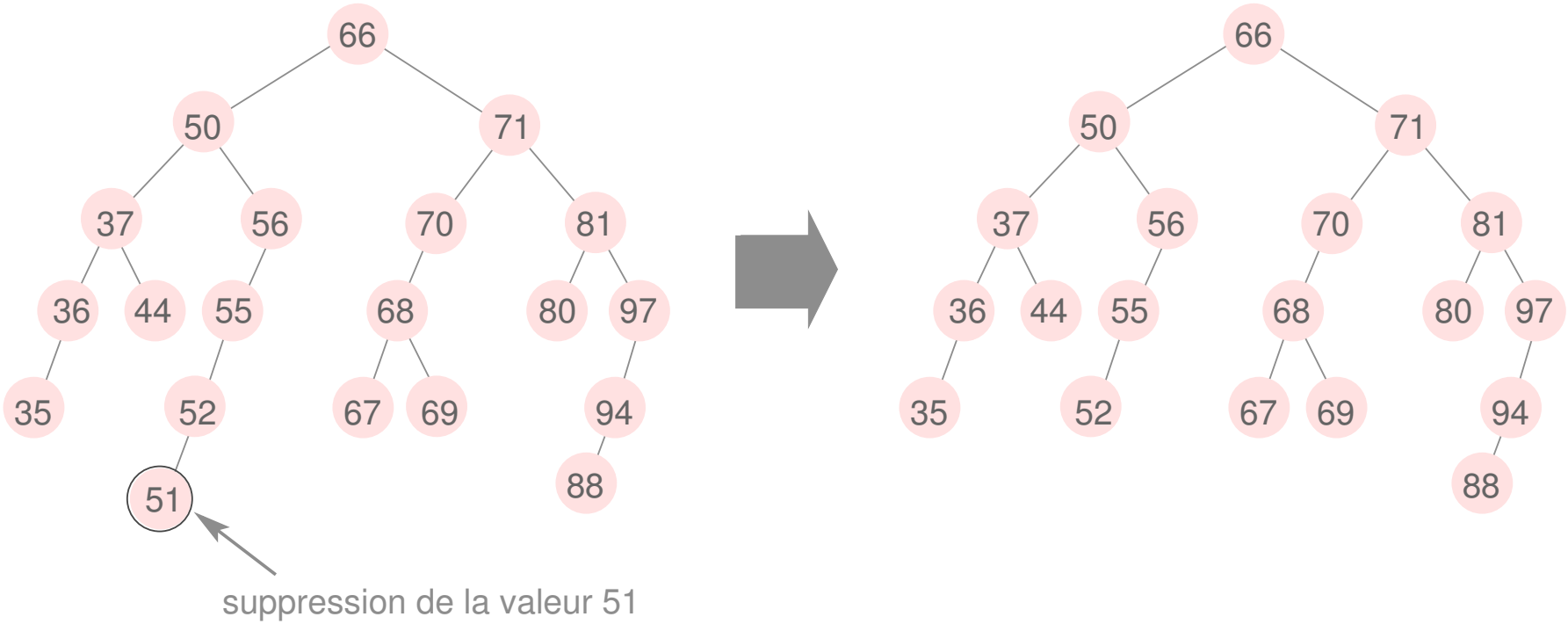




Complexité : $O(hauteur)$

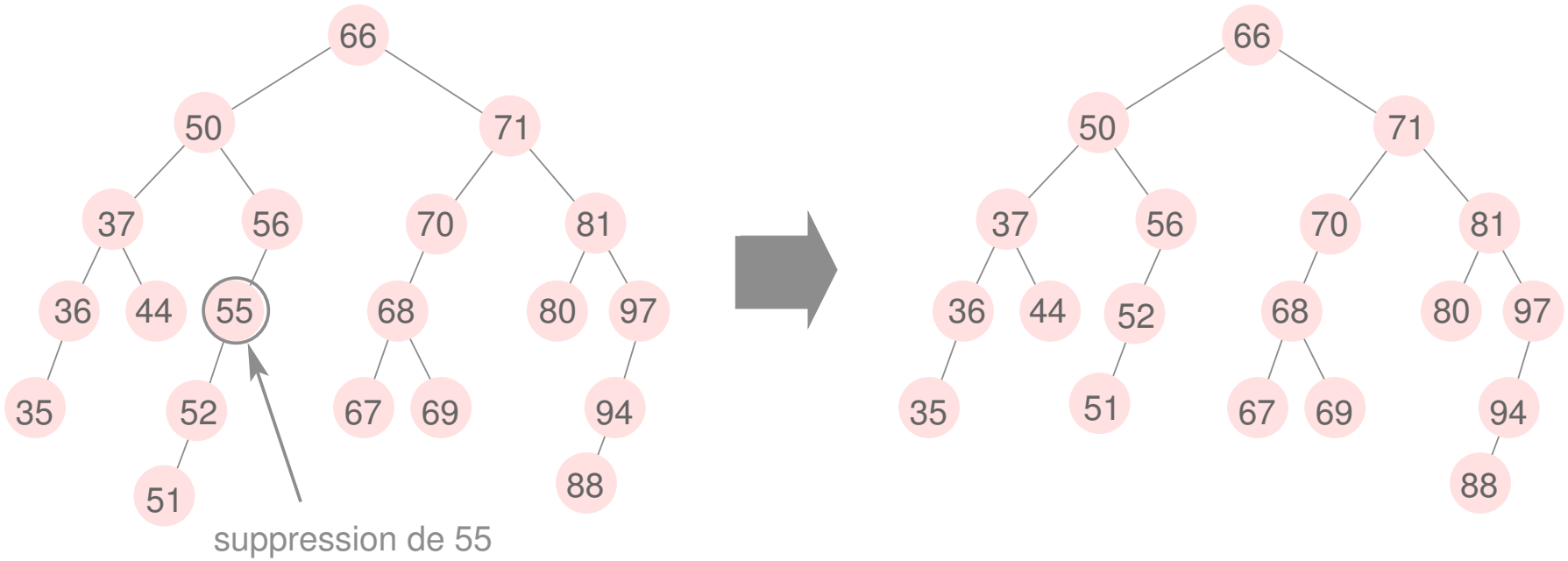
Suppression d'un élément dans un ABR

1. Recherche puis, suppression dans le cas où l'élément est dans une feuille



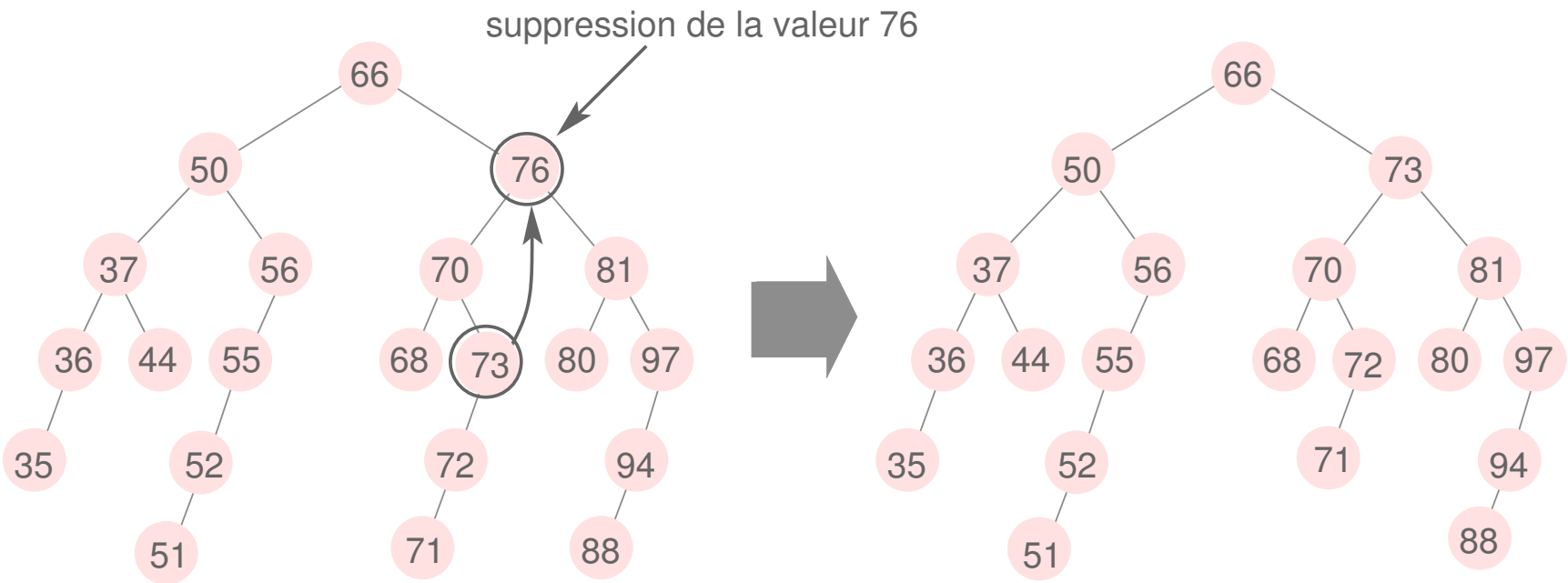
Suppression d'un élément dans un ABR

2. Recherche puis, compression dans le cas où l'élément est un nœud unaire



Suppression d'un élément dans un ABR

3. Recherche puis, remplacement par le maximum du sous-arbre gauche, dans le cas où l'élément est un nœud binaire



Suppression d'un élément dans un ABR

1. Recherche puis, suppression dans le cas où l'élément est dans une feuille
2. Recherche puis, compression dans le cas où l'élément est un nœud unaire
3. Recherche puis, remplacement par le maximum du sous-arbre gauche, dans le cas où l'élément est un nœud binaire

Suppression d'un élément dans un ABR

1. Recherche puis, suppression dans le cas où l'élément est dans une feuille
2. Recherche puis, compression dans le cas où l'élément est un nœud unaire
3. Recherche puis, remplacement par le maximum du sous-arbre gauche, dans le cas où l'élément est un nœud binaire

Complexité : dans tous les cas, $O(hauteur)$

Test du vide de l'ensemble dans un ABR

- Tester si l'arbre est vide ou non

Test du vide de l'ensemble dans un ABR

- Tester si l'arbre est vide ou non

Complexité : $O(1)$

Hauteur d'un ABR ?

On a déjà vu que $\lceil \log_2(n+1) \rceil \leq \textit{hauteur} \leq n$

Hauteur d'un ABR ?

On a déjà vu que $\lceil \log_2(n+1) \rceil \leq \textit{hauteur} \leq n$

Dans le *meilleur des cas*, complexités des opérations dans les ABR : $O(\log(n))$

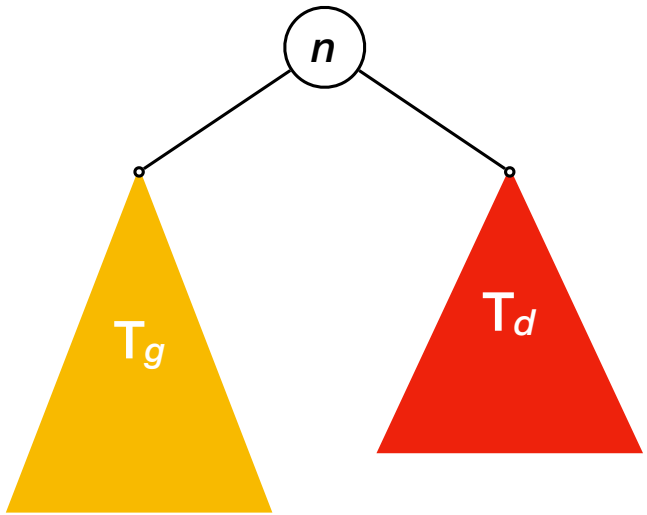
Dans le *pire des cas*, complexités des opérations dans les ABR : $O(n)$, pire que l'utilisation de tableaux triés

Comment maîtriser la hauteur d'un ABR ?

De nombreuses méthodes existent : arbres rouge-noir, arbres fortement équilibrés, **arbres AVL**, arbres 2-3, arbres 2-3-4, arbres B...

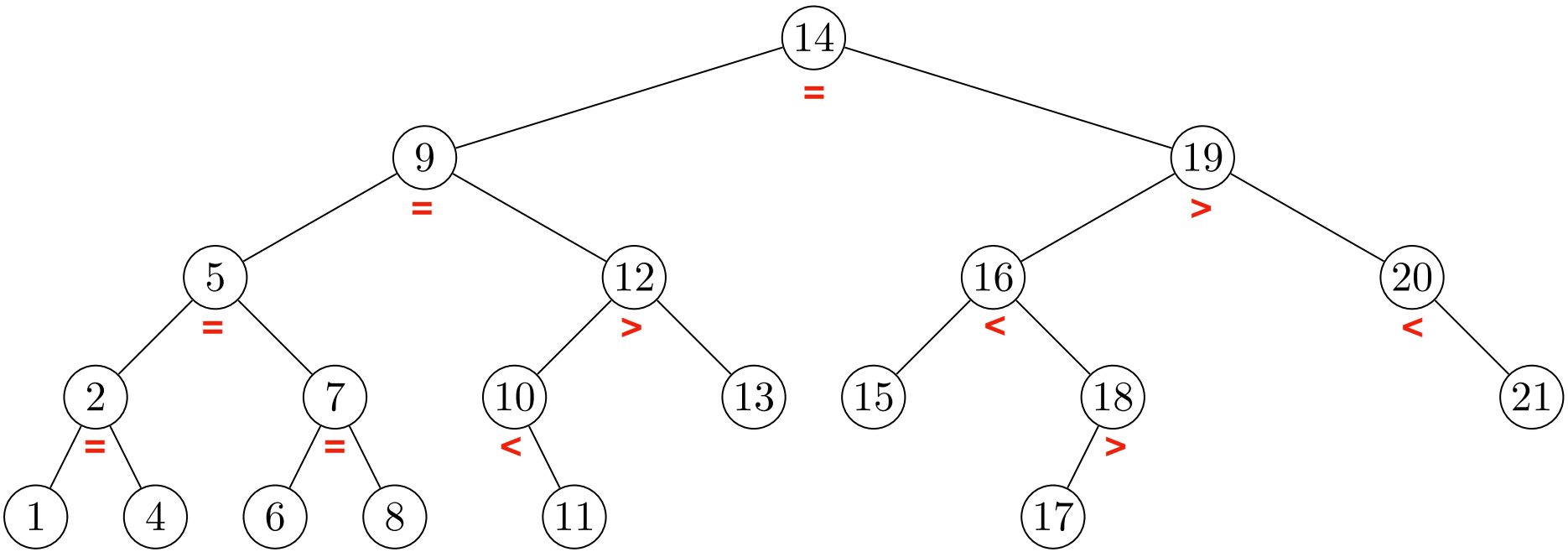
- Arbres AVL (Adelson-Velskii et Landis):

pour tout sous-arbre :



$$|h(T_g) - h(T_d)| \leq 1$$

Exemple d'AVL



Majoration de la hauteur d'un AVL

La hauteur d'un arbre AVL à n nœuds est inférieure ou égale à $1,45 \log_2 n$.

Majoration de la hauteur d'un AVL

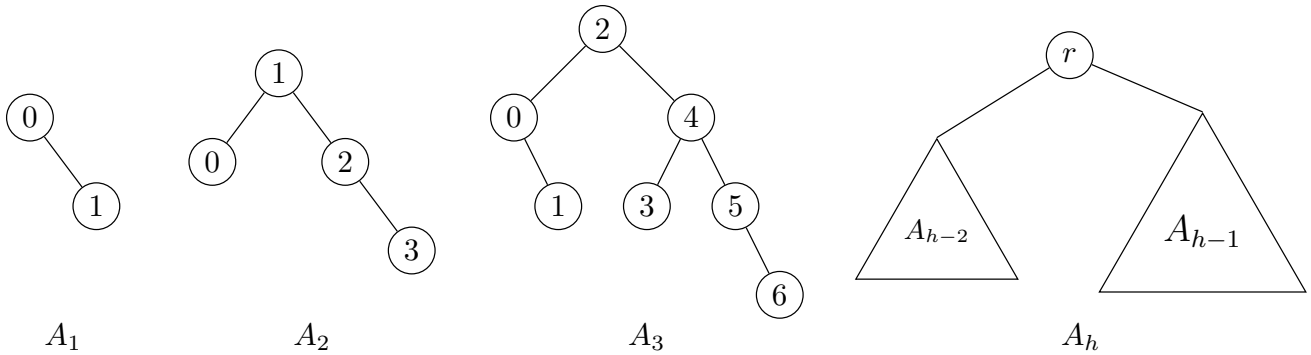
La hauteur d'un arbre AVL à n nœuds est inférieure ou égale à $1,45 \log_2 n$.

Preuve : Minorer le nombre n_h de nœud d'un arbre AVL de hauteur h...

Majoration de la hauteur d'un AVL

La hauteur d'un arbre AVL à n nœuds est inférieure ou égale à $1,45 \log_2 n$.

Preuve : Minorer le nombre n_h de nœud d'un arbre AVL de hauteur h...



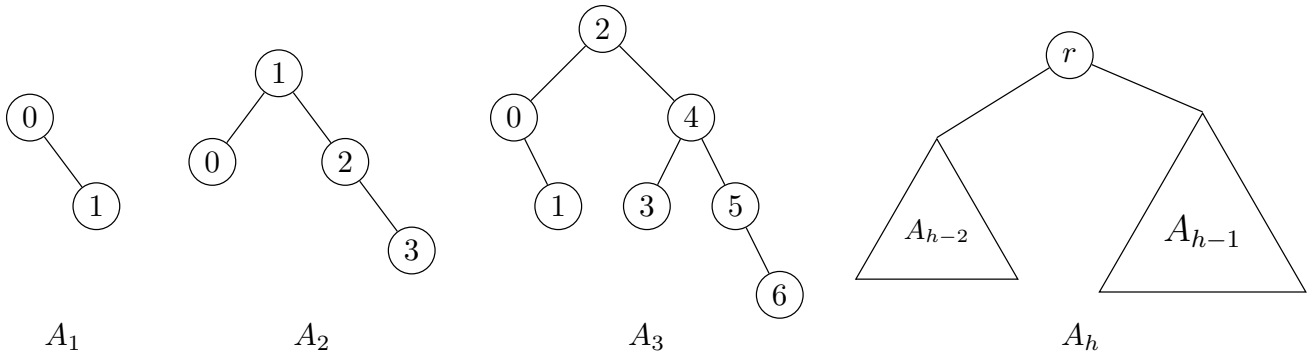
Arbres de Fibonacci

$$n_h = 1 + n_{h-2} + n_{h-1} \text{ avec } n_0 = 1,\ n_1 = 2$$

Majoration de la hauteur d'un AVL

La hauteur d'un arbre AVL à n nœuds est inférieure ou égale à $1,45 \log_2 n$.

Preuve : Minorer le nombre n_h de nœud d'un arbre AVL de hauteur h...



Arbres de Fibonacci

$$n_h = 1 + n_{h-2} + n_{h-1} \text{ avec } n_0 = 1,\ n_1 = 2$$

Résolution...

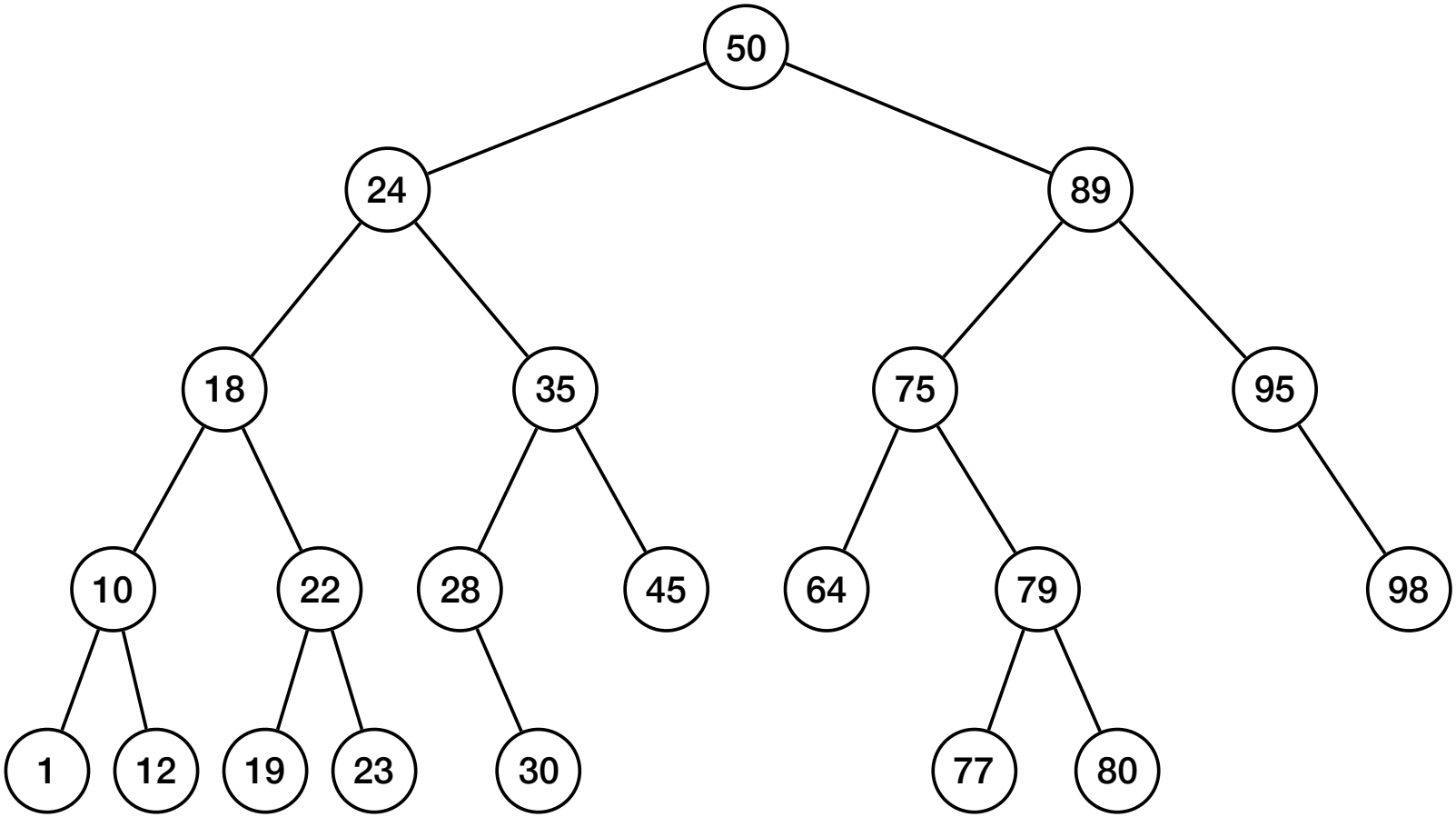
$$n_h = \left(1 + \frac{2}{\sqrt{5}}\right) \cdot \left(\frac{1+\sqrt{5}}{2}\right)^h + \left(1 - \frac{2}{\sqrt{5}}\right) \cdot \left(\frac{1-\sqrt{5}}{2}\right)^h - 1 \geq \left(1 + \frac{2}{\sqrt{5}}\right) \cdot \left(\frac{1+\sqrt{5}}{2}\right)^h$$

$$\text{d'où } \log_2 n_h \geq h \log_2 \left(\frac{1+\sqrt{5}}{2}\right)$$

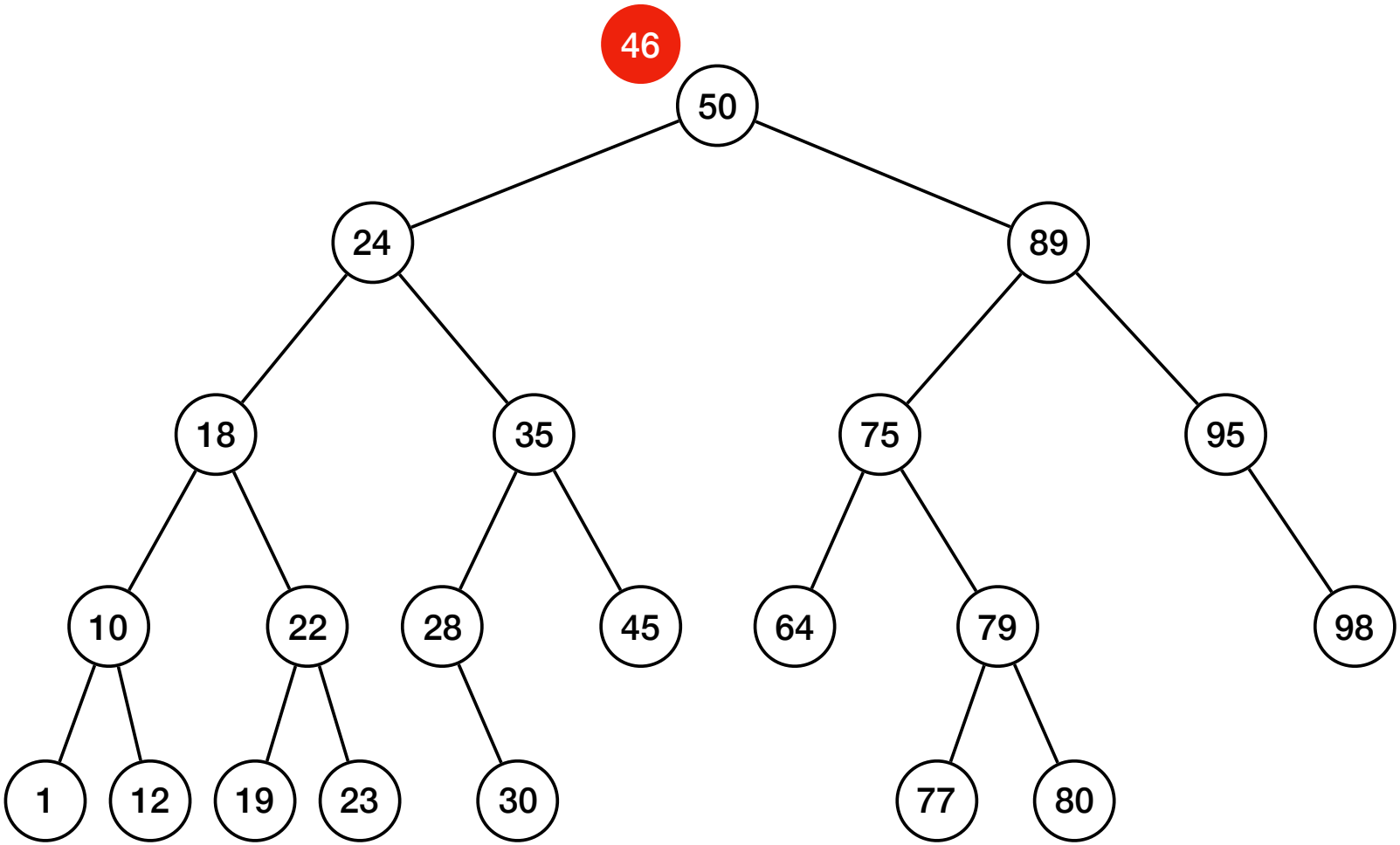
Test d'appartenance dans un AVL à n nœuds

Complexité dans le pire des cas : $O(\log(n))$

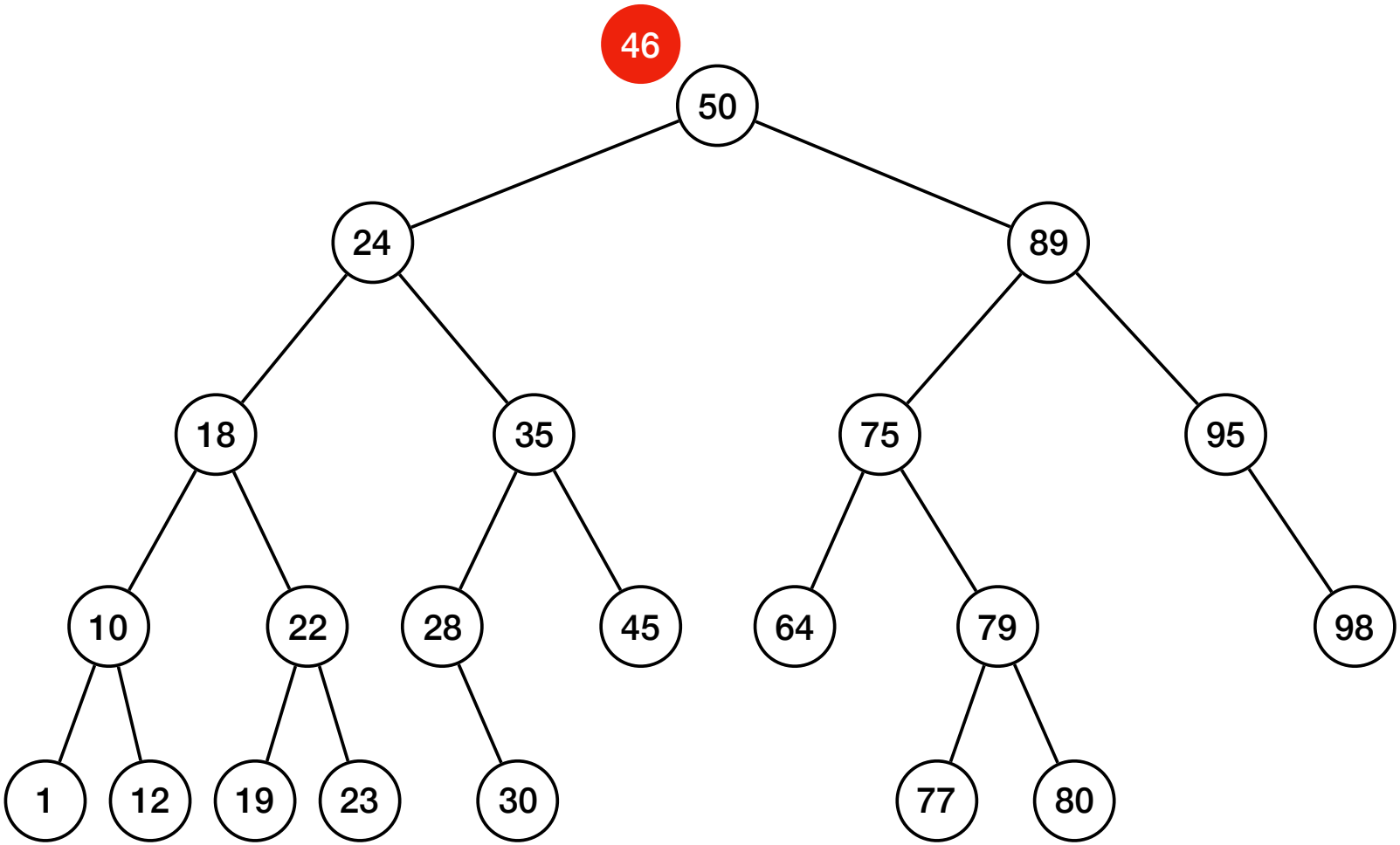
Comment maintenir la propriété AVL au cours d'un insertion ou d'une suppression ?



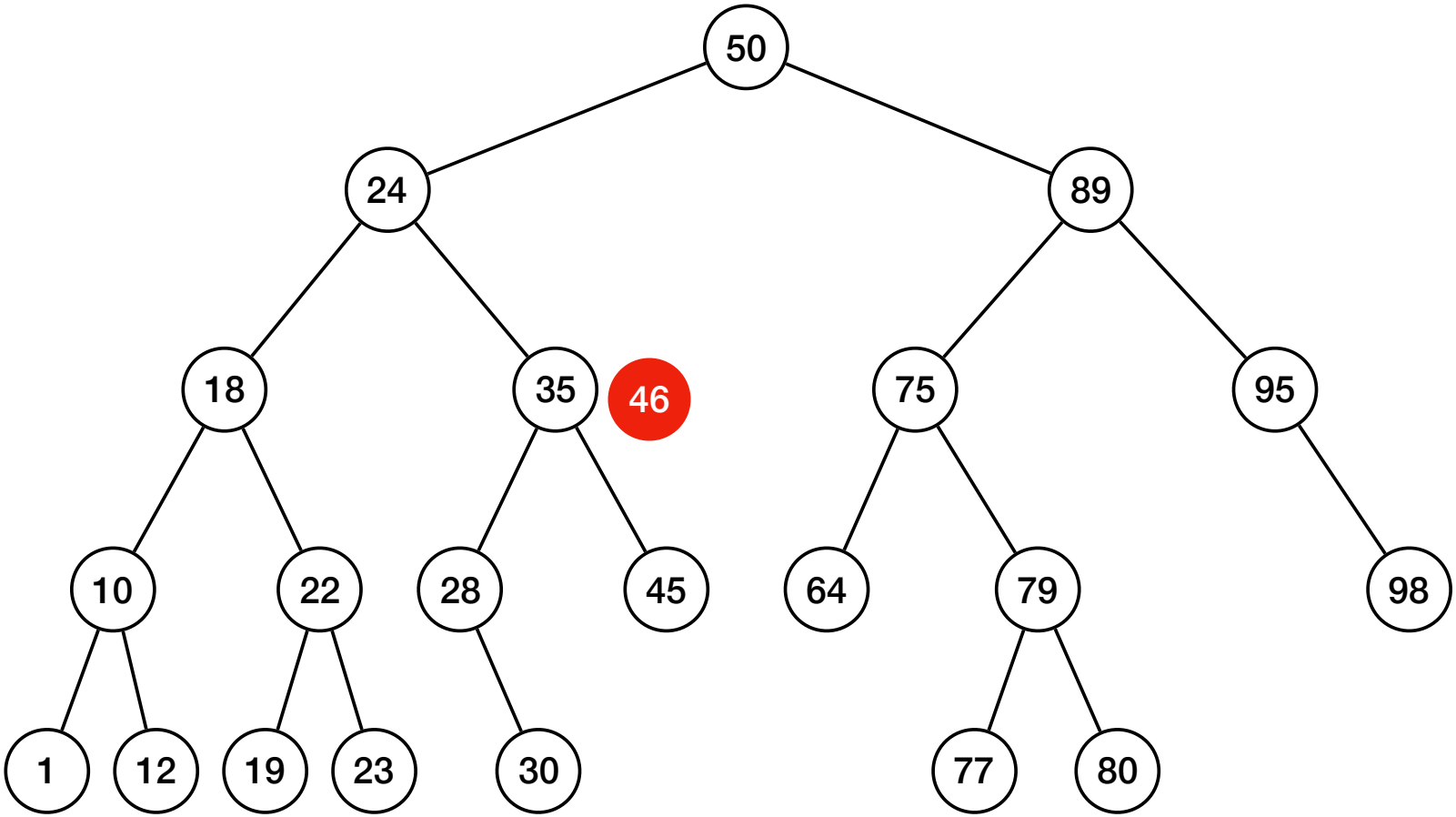
Insertion



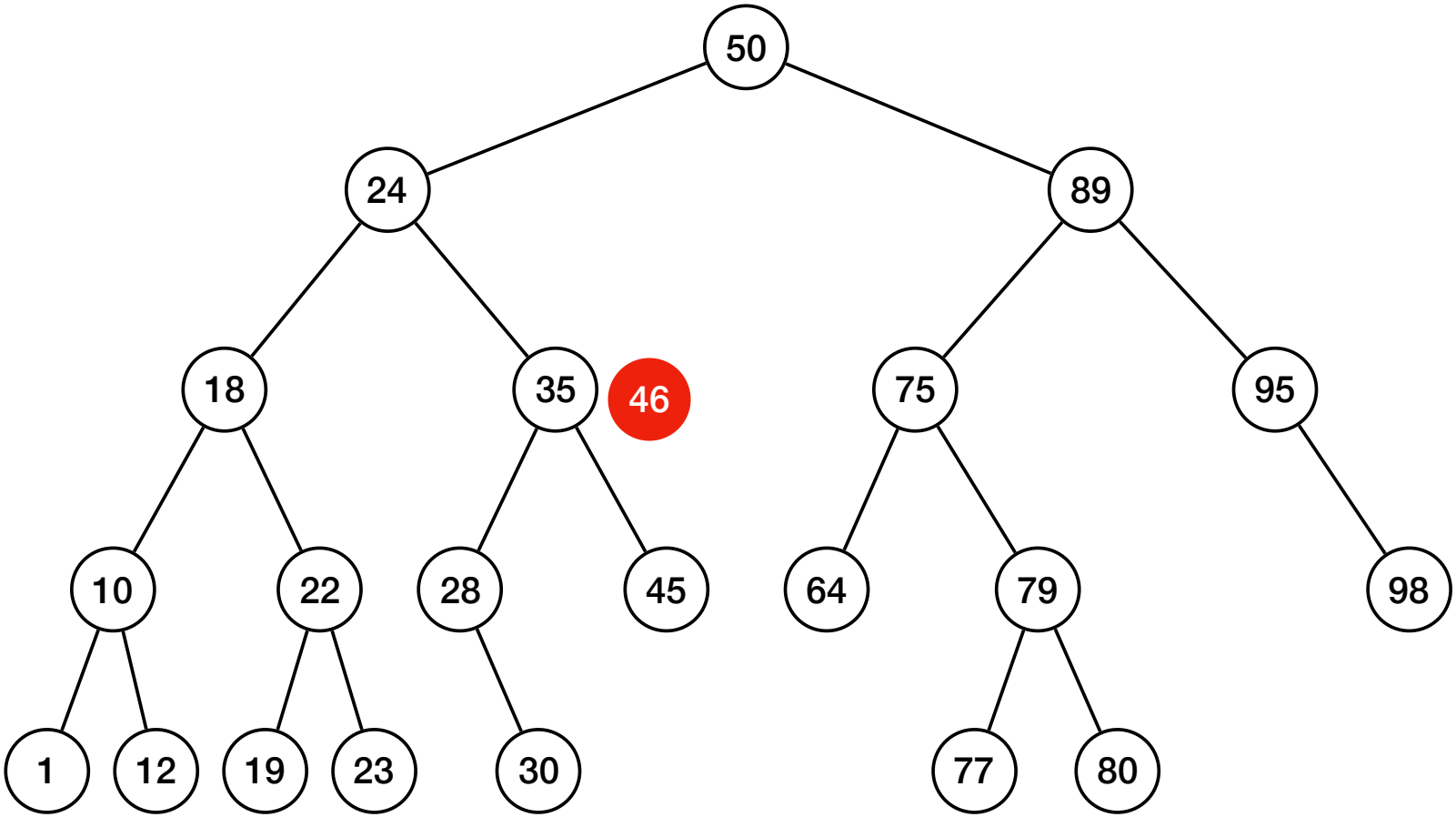
Insertion



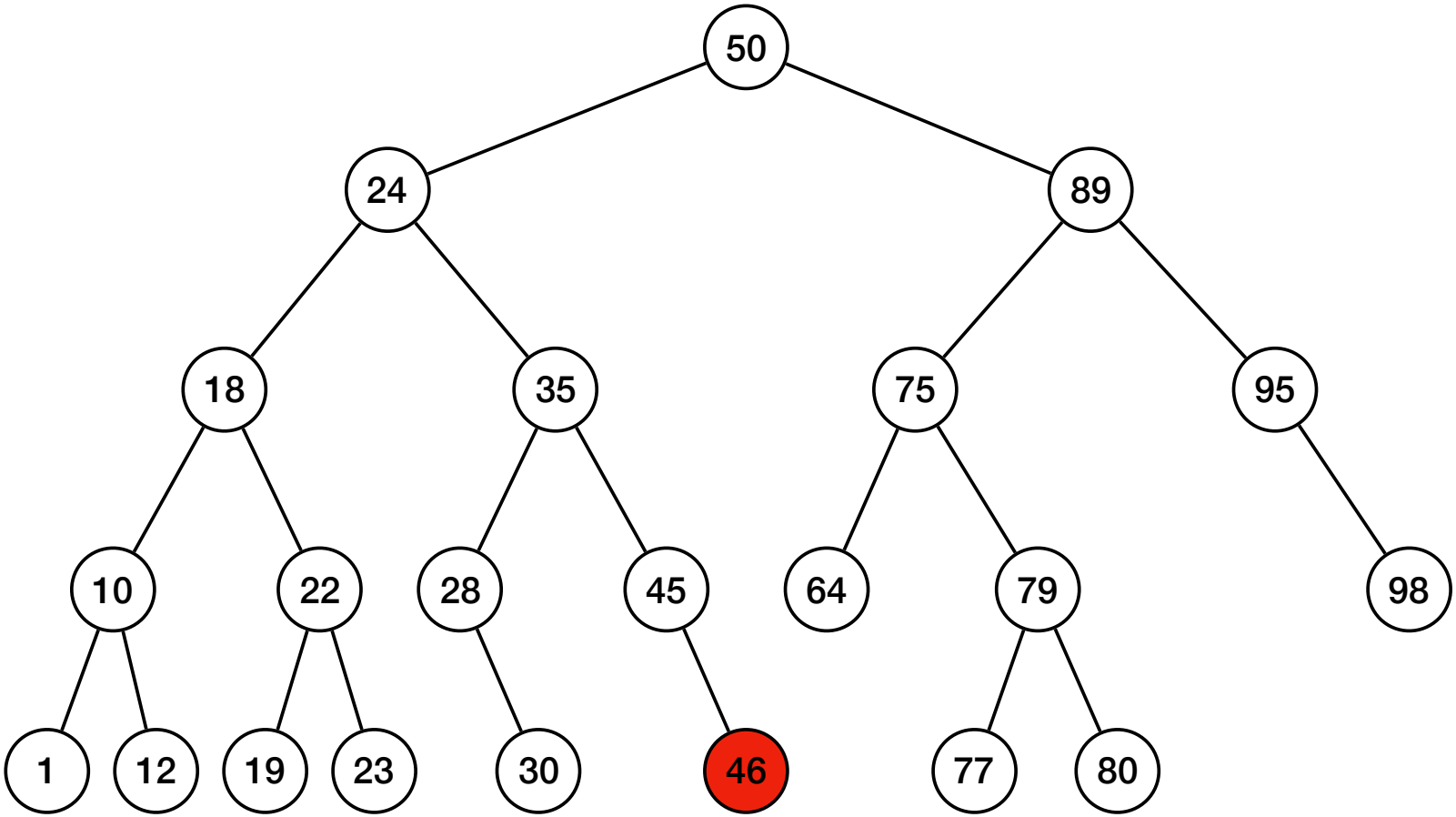
Insertion



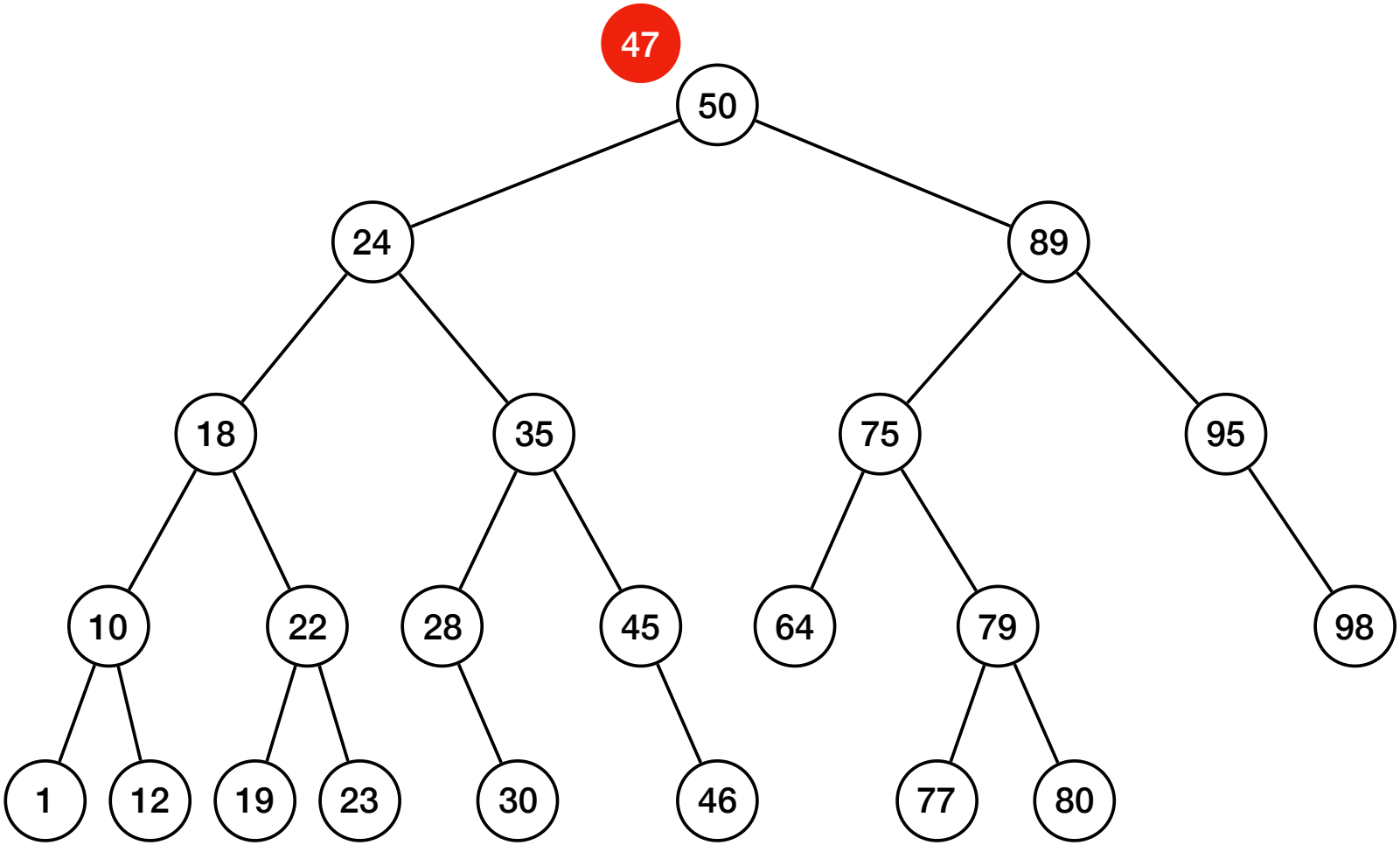
Insertion



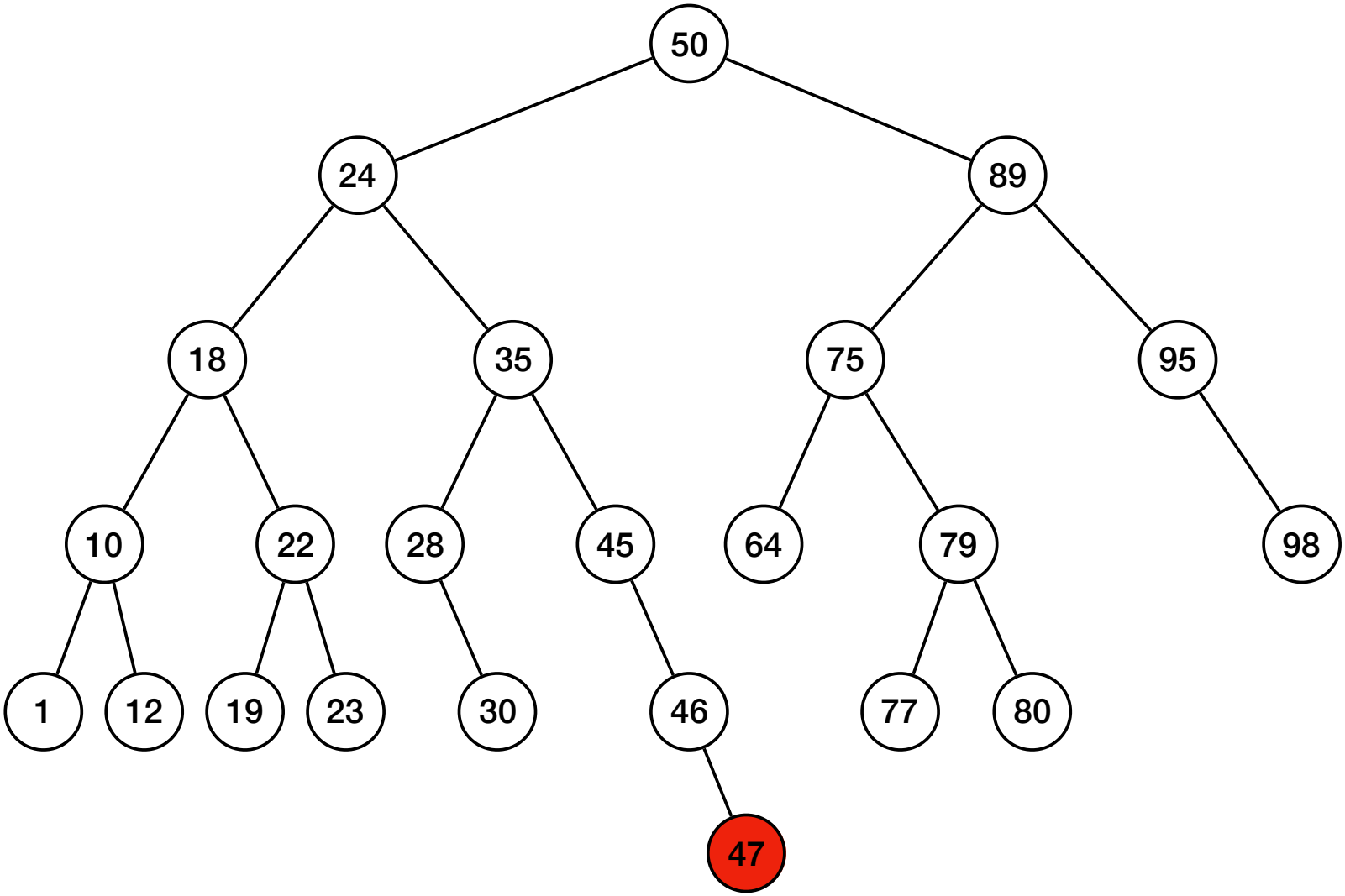
Insertion



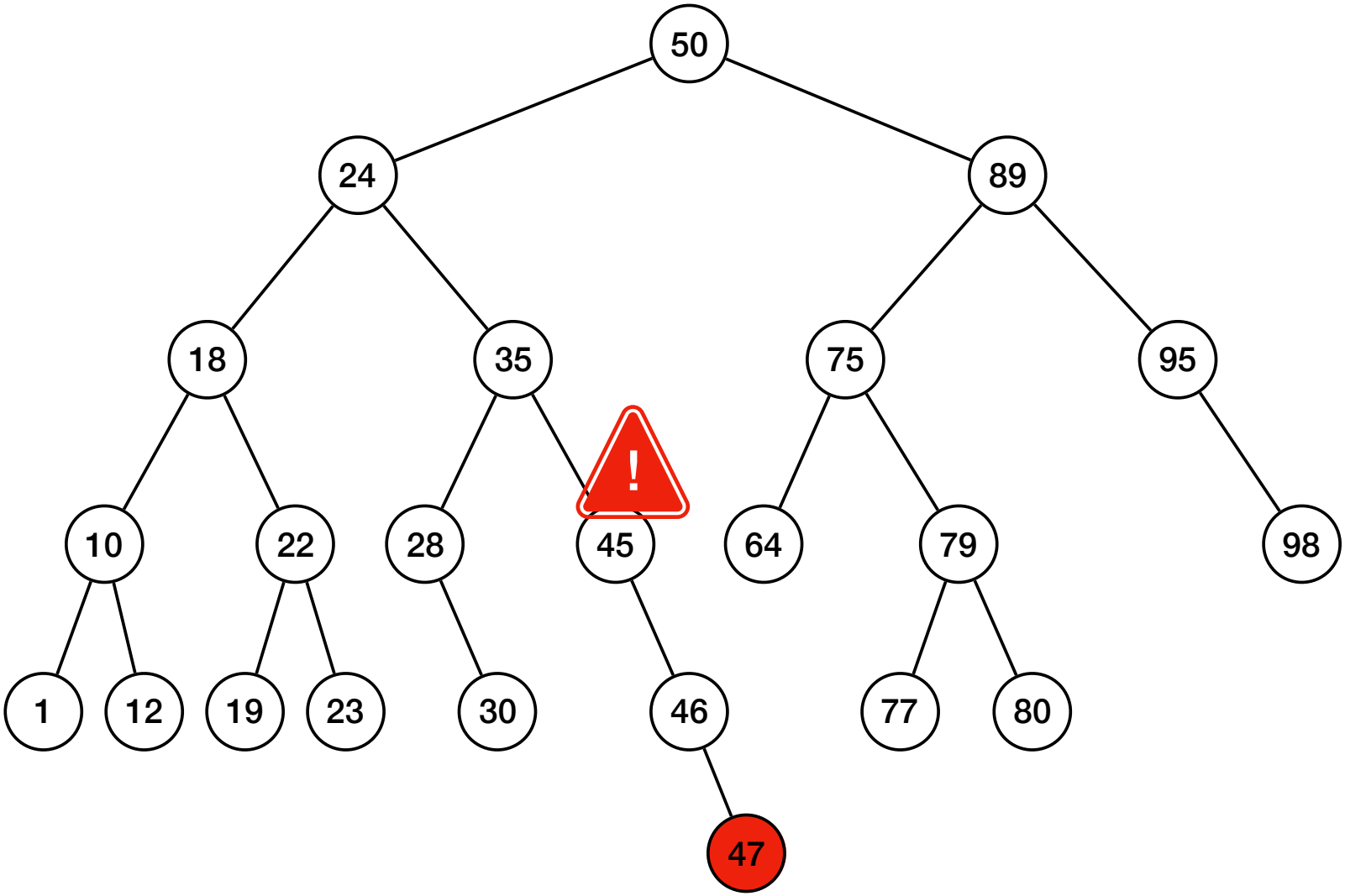
Insertion



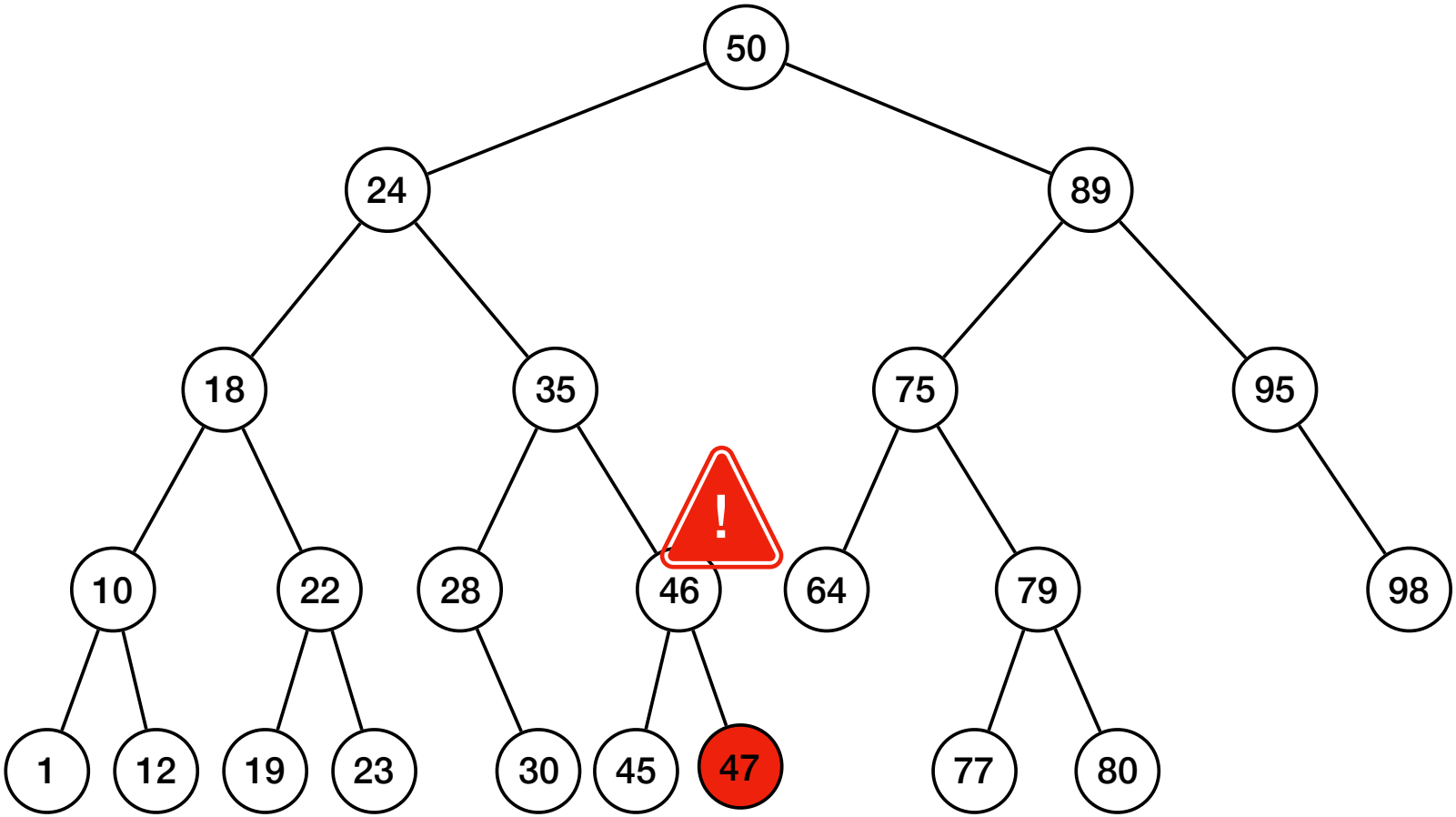
Insertion



Insertion

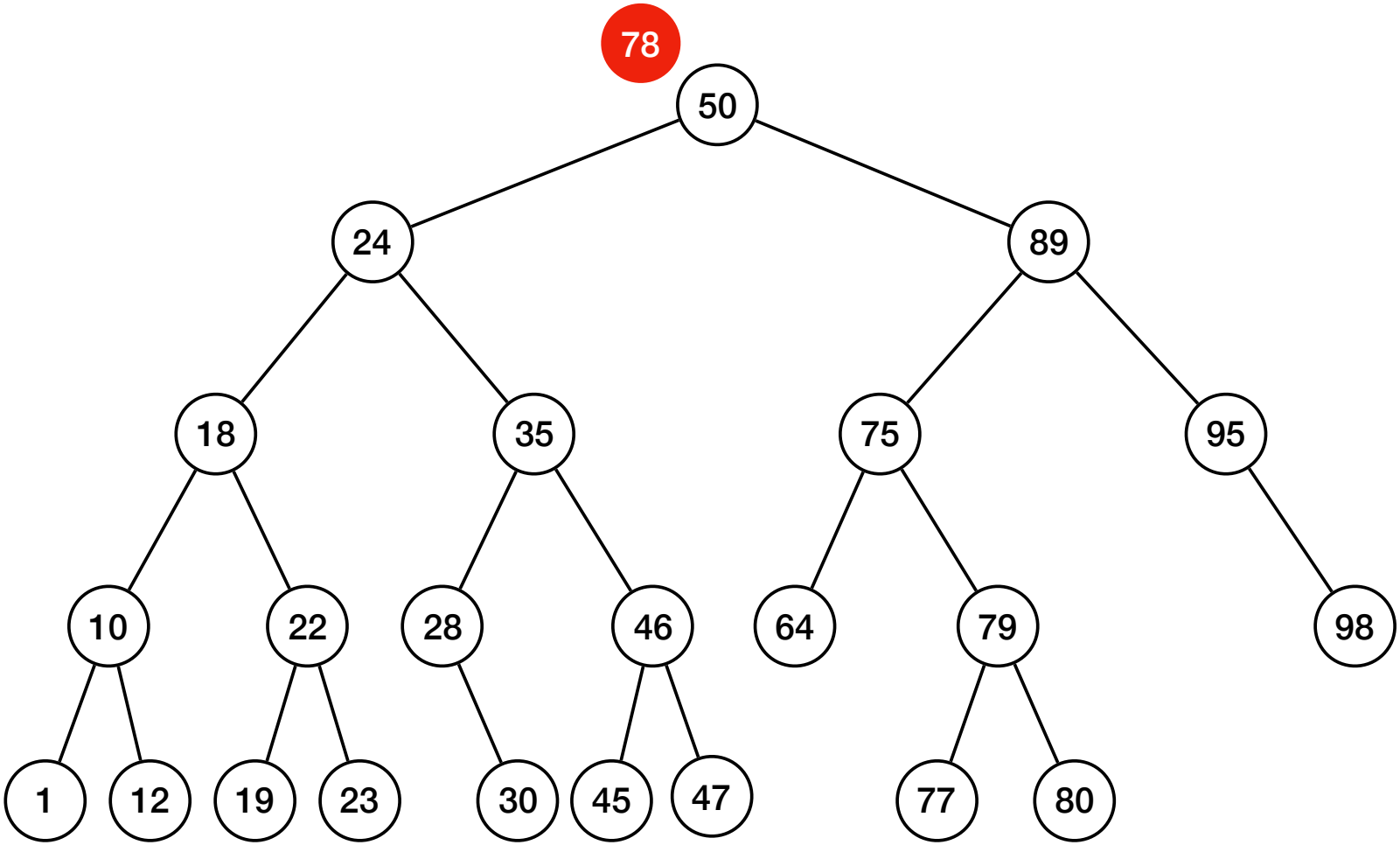


Insertion

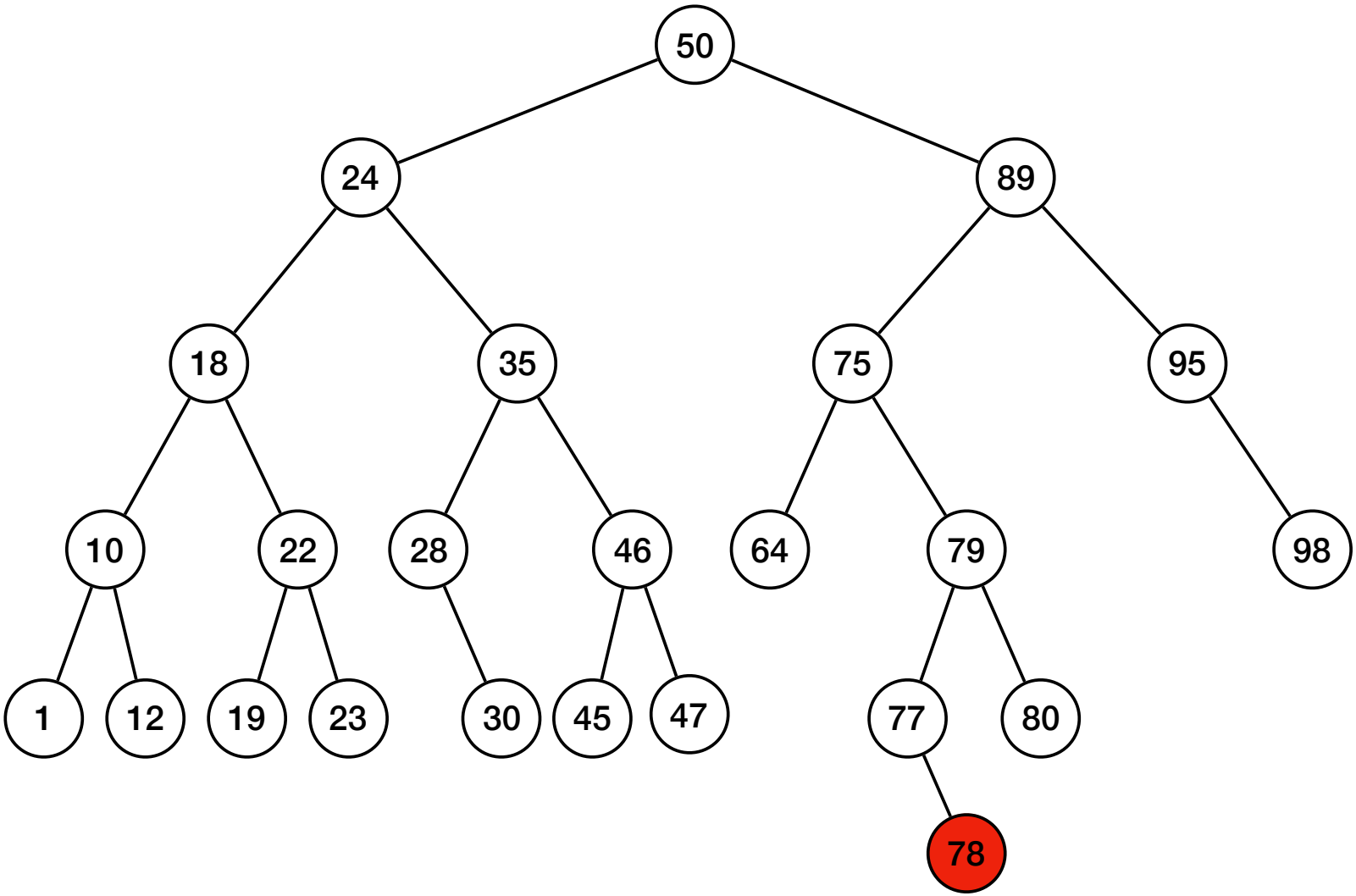


Rotation simple

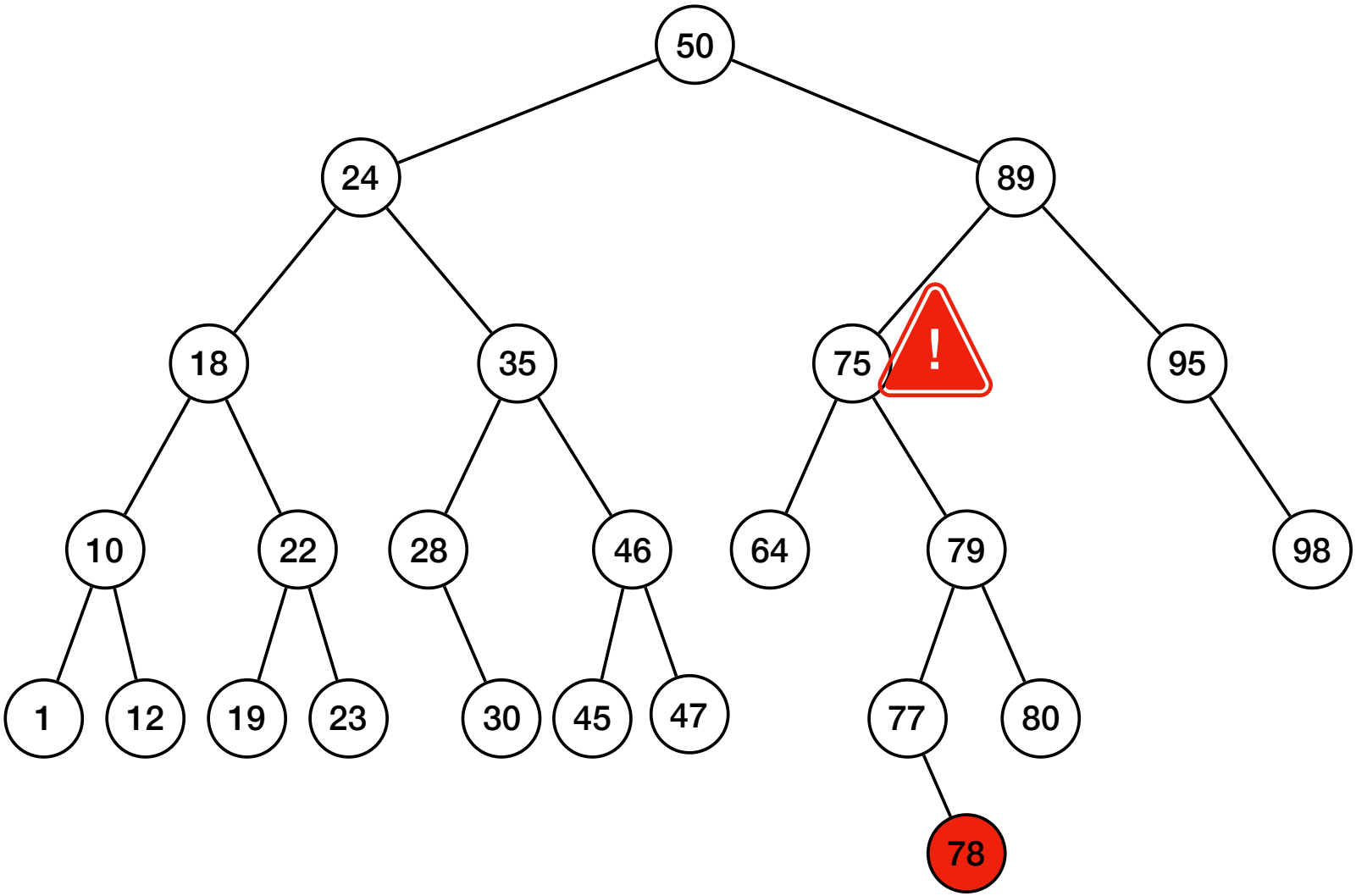
Insertion



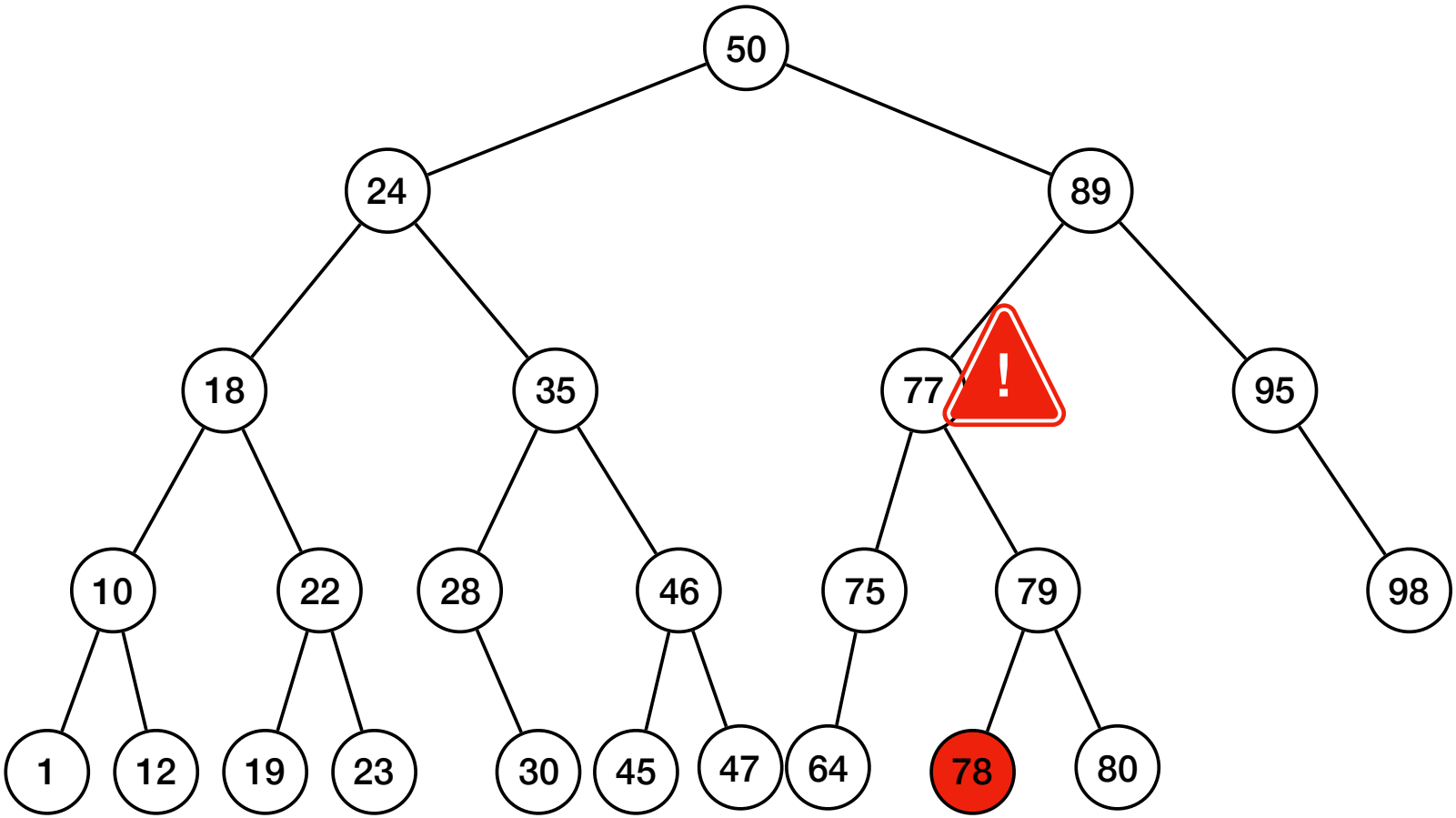
Insertion



Insertion

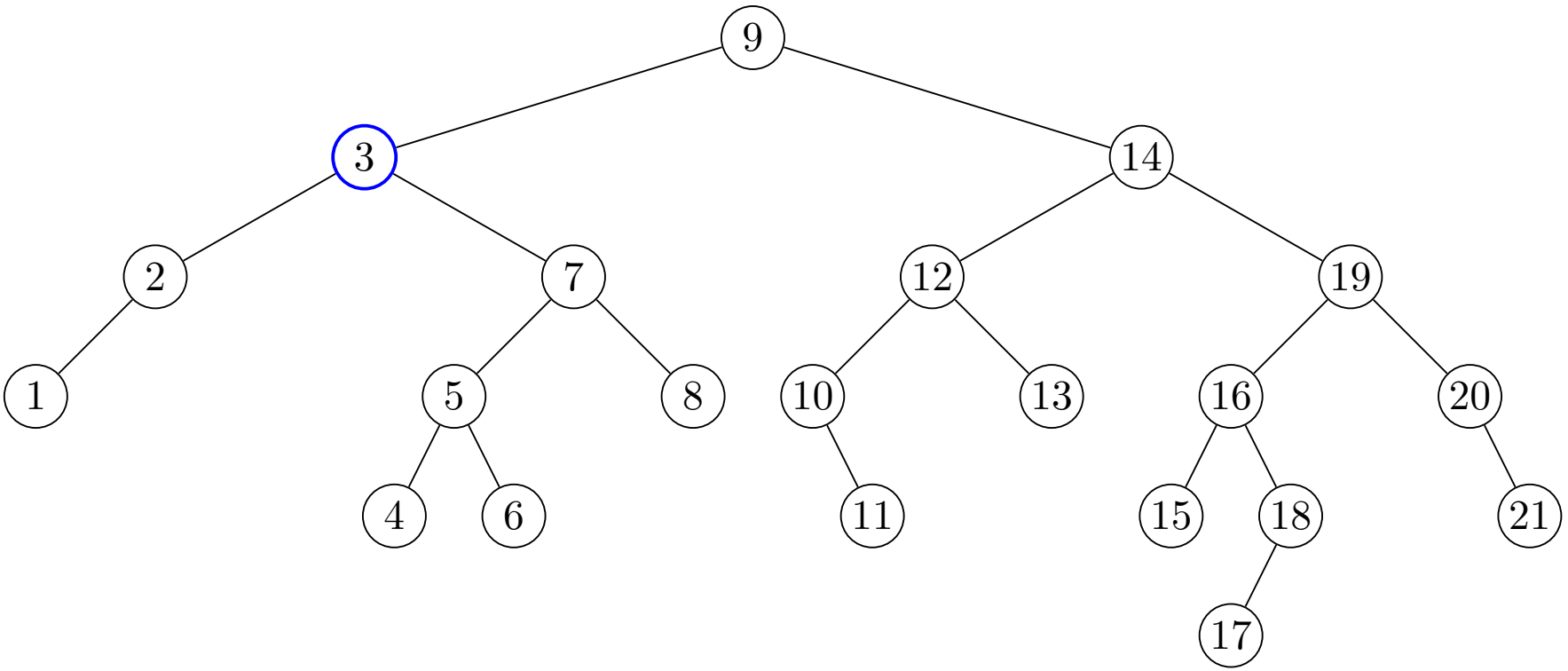


Insertion

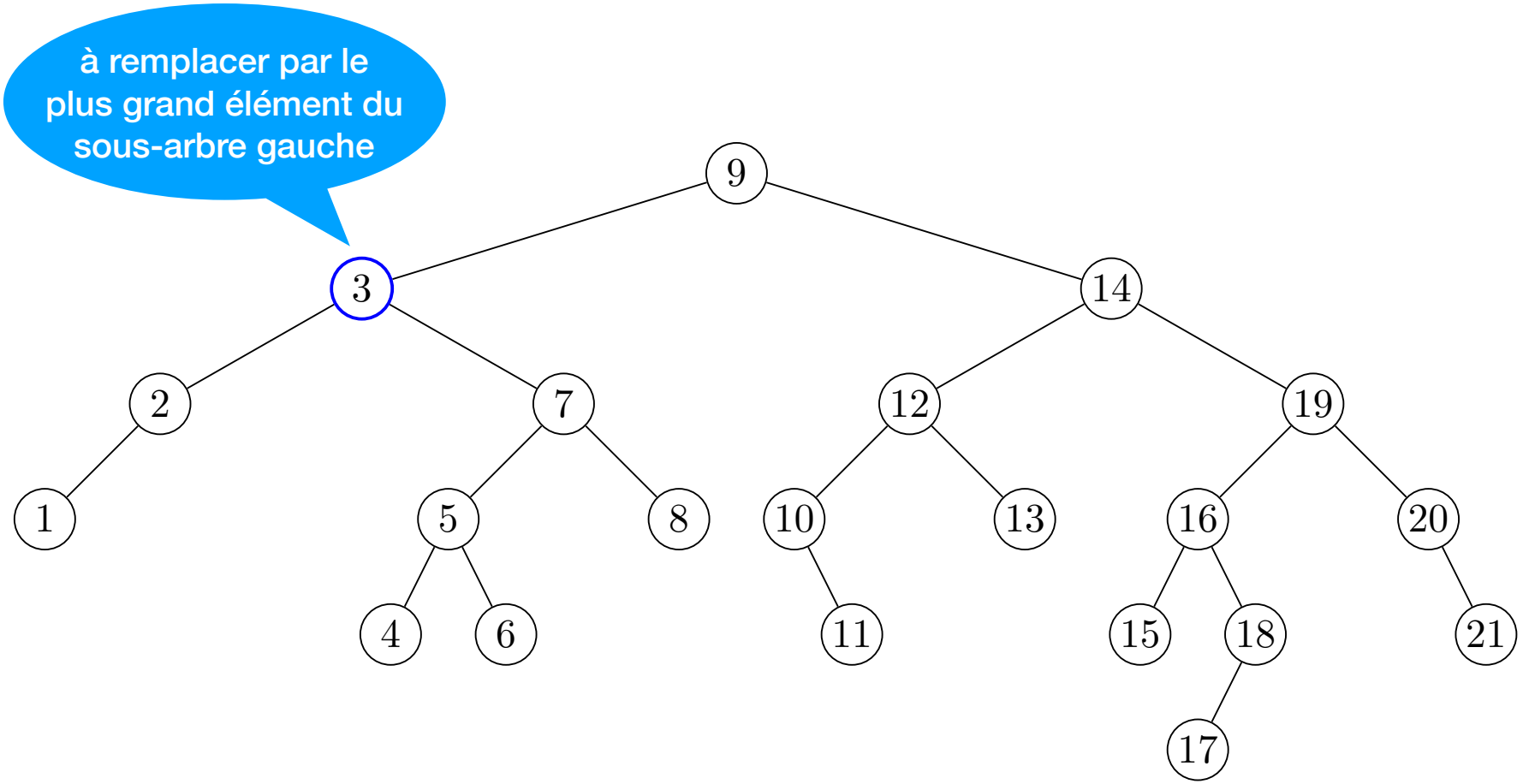


Rotation double

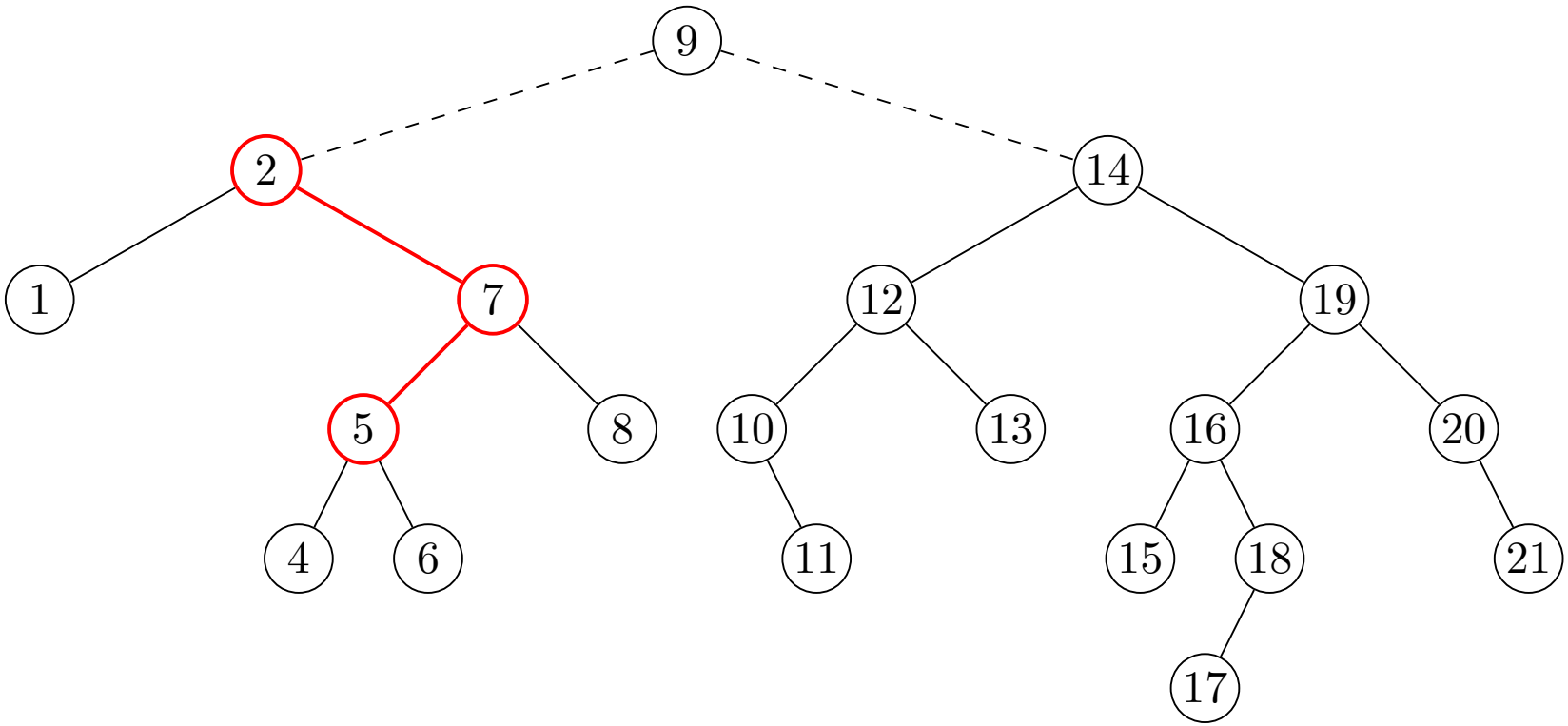
Suppression



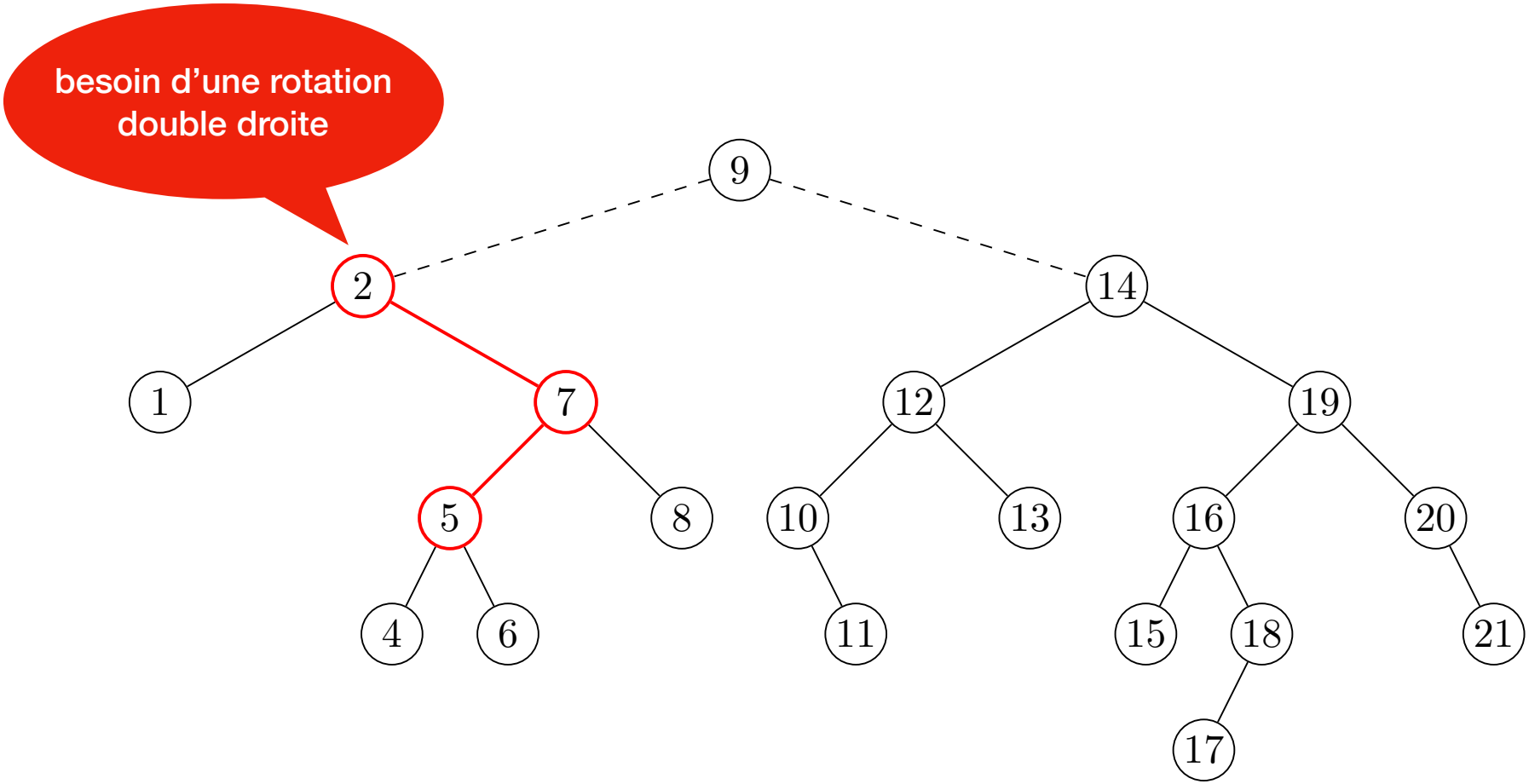
Suppression



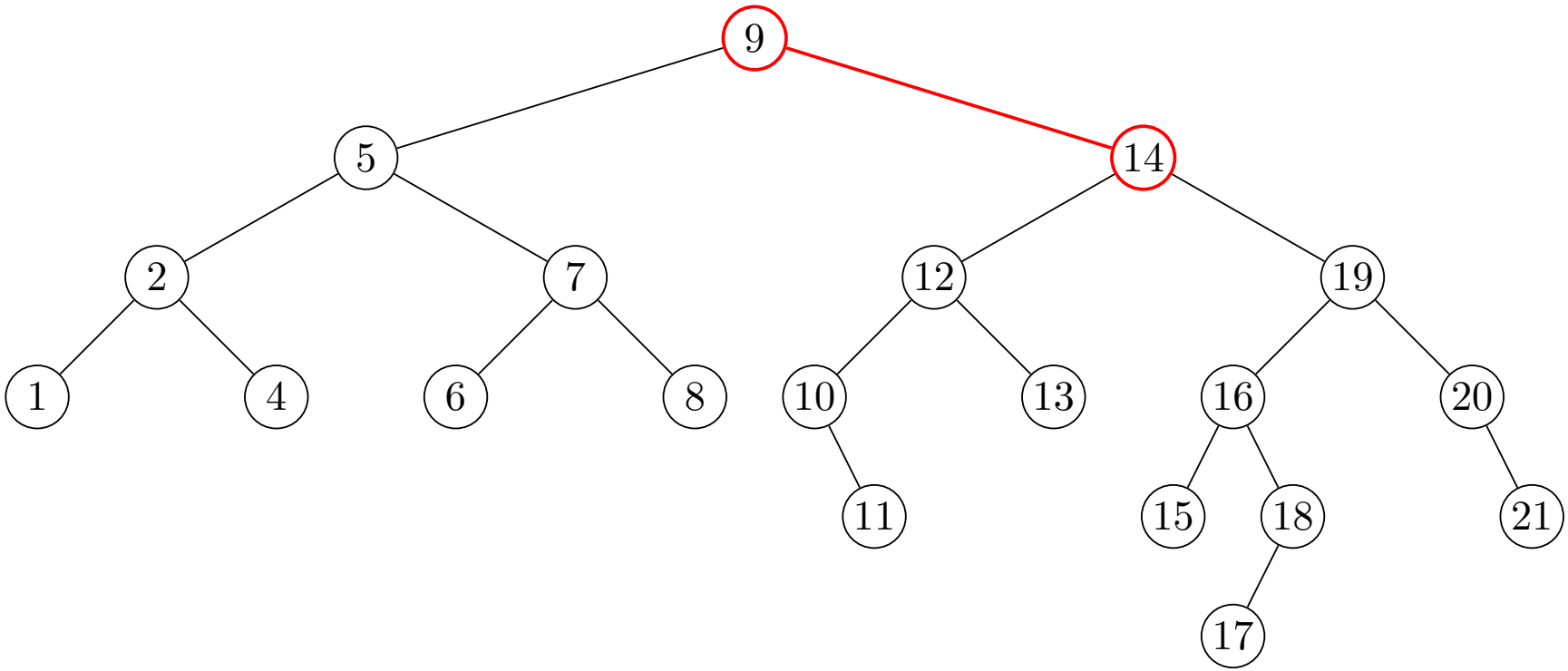
Suppression



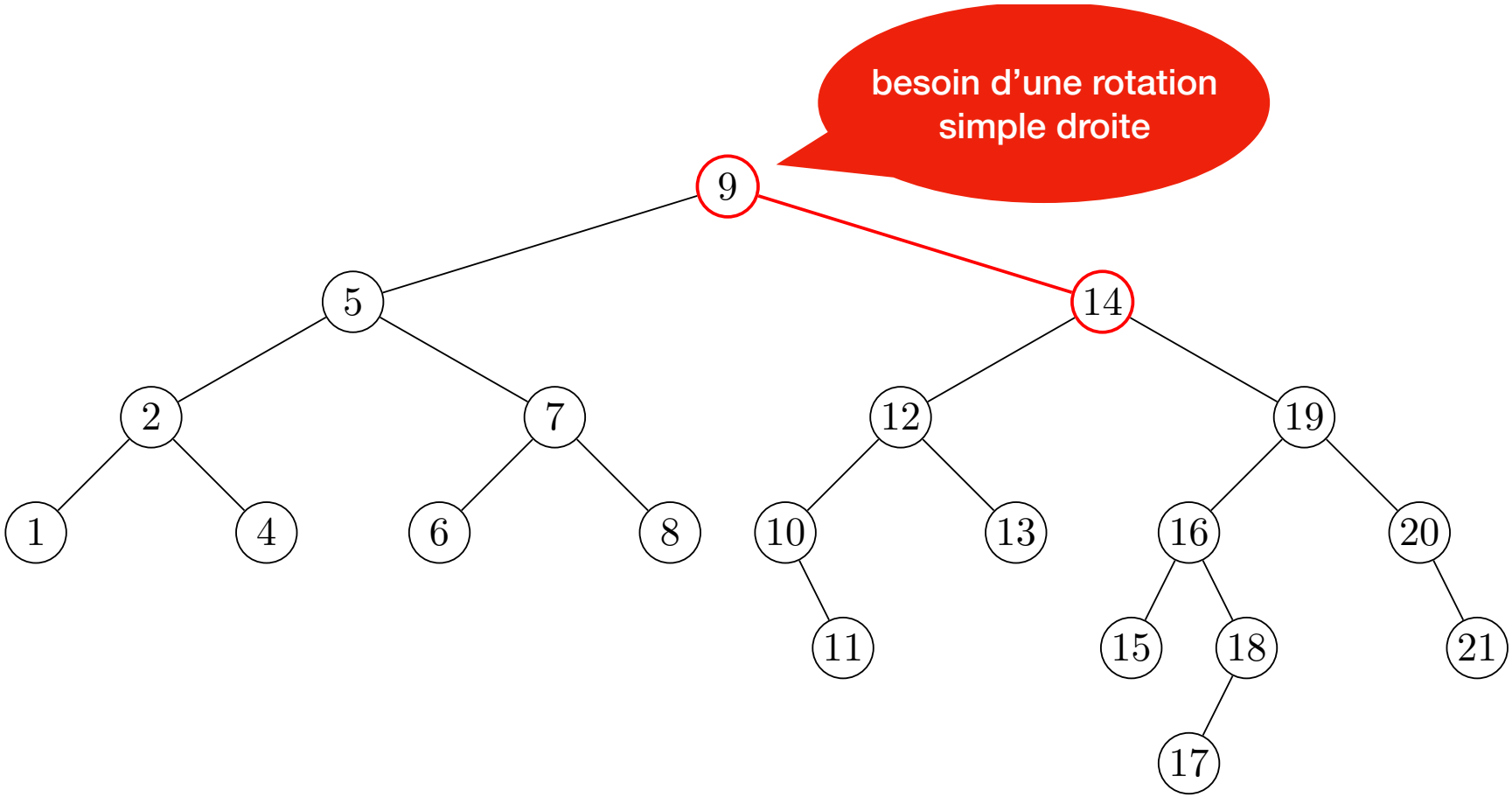
Suppression



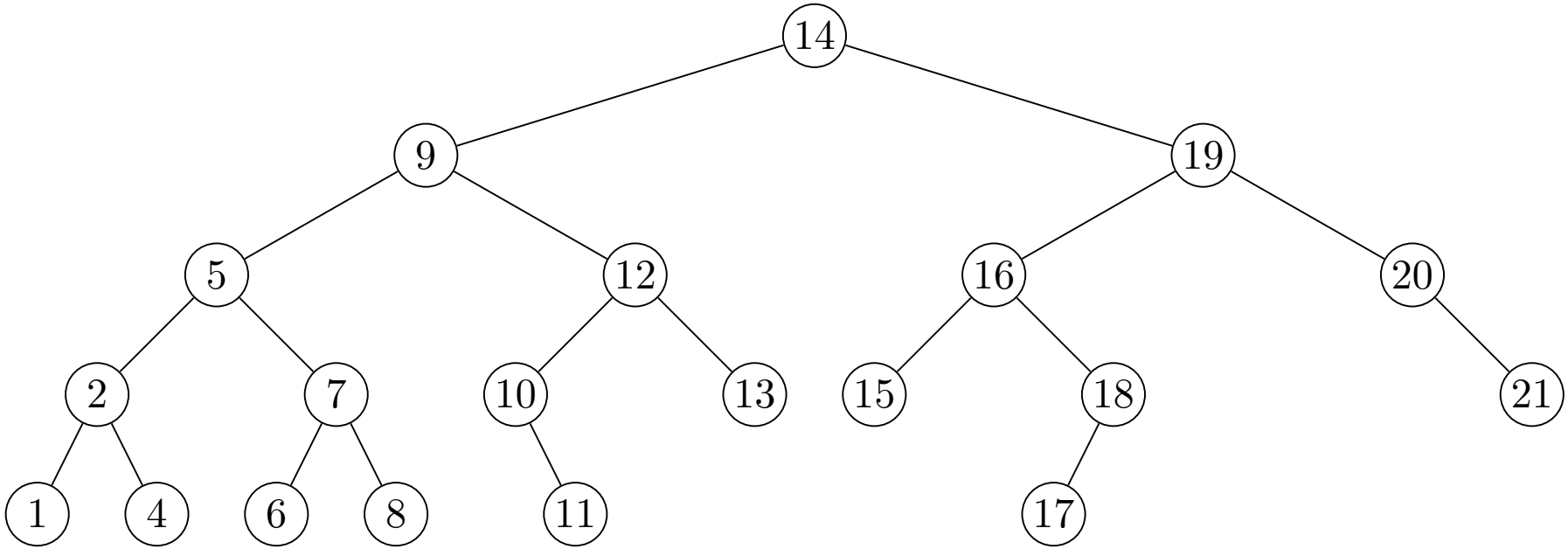
Suppression



Suppression



Suppression



Utilisation des AVL : résumé

Pour un AVL à n nœuds :

- Recherche d'un élément : $O(\log n)$
- Insertion d'un élément : $O(\log n)$
- Suppression d'un élément : $O(\log n)$

Utilisation des AVL : résumé

Pour un AVL à n nœuds :

- Recherche d'un élément : $O(\log n)$
- Insertion d'un élément : $O(\log n)$
- Suppression d'un élément : $O(\log n)$

On ne peut pas faire mieux... si la seule opération autorisée sur les clés est la comparaison de deux clés (pour faire mieux, il faut savoir hacher par exemple)