



LSINF1121: Bilan 3 Arbres de recherche

Pierre Schaus

Vrai ou faux ?

- Dans le meilleur des cas, le nombre de comparaisons entre clefs pour une recherche binaire d'une clef particulière dans un tableau trié de N clefs distinctes est $\sim \log(N)$?
- Faux: Cela correspond au pire-cas, dans le meilleur des cas on tombe directement sur la clef donc $O(1)$.

Rappel: algorithmes pour traverser un arbre

```
visiterPréfixe(Arbre A) :  
    visiter (A)  
    Si nonVide (gauche(A))  
        visiterPréfixe(gauche(A))  
    Si nonVide (droite(A))  
        visiterPréfixe(droite(A))
```

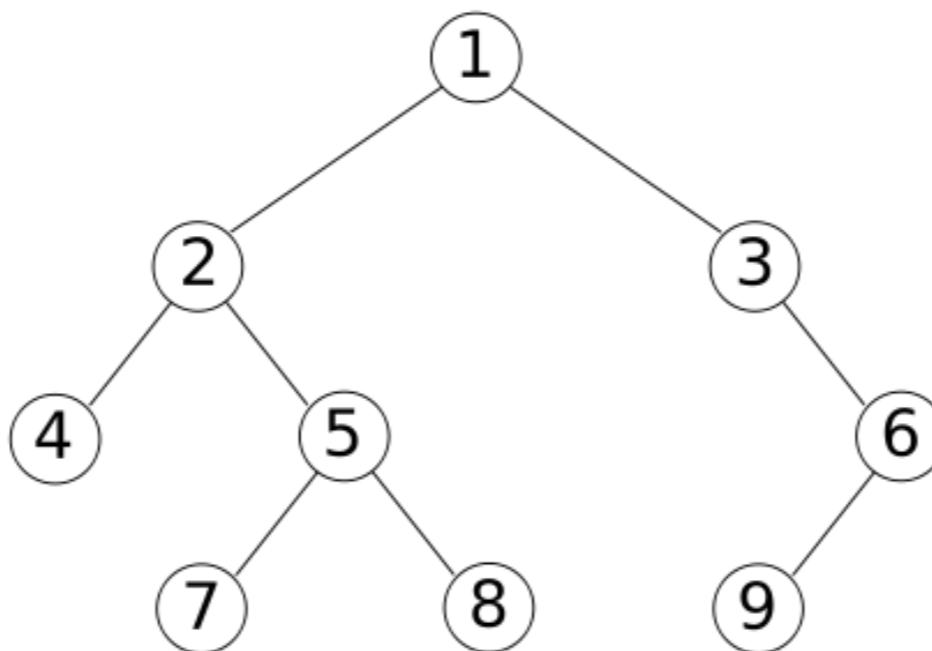
```
visiterPostfixe(Arbre A) :  
    Si nonVide(gauche(A))  
        visiterPostfixe(gauche(A))  
    Si nonVide(droite(A))  
        visiterPostfixe(droite(A))  
    visiter(A)
```

```
visiterInfixe(Arbre A) :  
    Si nonVide(gauche(A))  
        visiterInfixe(gauche(A))  
    visiter(A)  
    Si nonVide(droite(A))  
        visiterInfixe(droite(A))
```

1, 2, 4, 5, 7, 8, 3, 6, 9

4, 7, 8, 5, 2, 9, 6, 3, 1

4, 2, 7, 5, 8, 1, 3, 9, 6



!!Not a BST!!

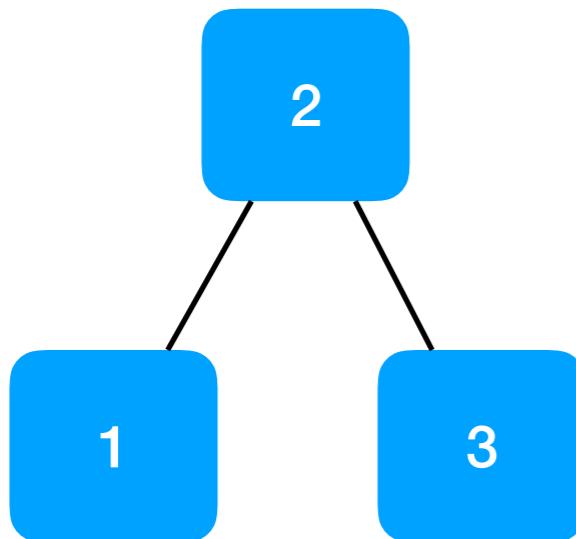
Vrai ou Faux ?

- Étant donné un parcours infixé d'un BST contenant N clefs distinctes.
 - Est-il possible de reconstruire la forme du BST sur base du résultat du parcours ?
 - Si oui, écrivez le pseudo-code d'un algorithme pour le faire, si non, donnez un contre-exemple qui justifie votre réponse

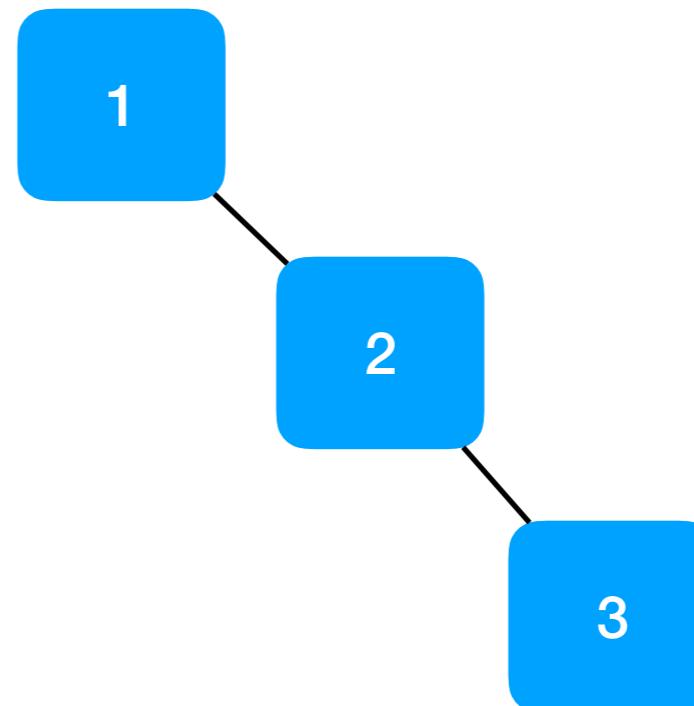
Contre exemple

- $\text{infixe}(T1) = \text{infixe}(T2) = 1-2-3$ et $T1 \neq T2$

- $T1$



- $T2$



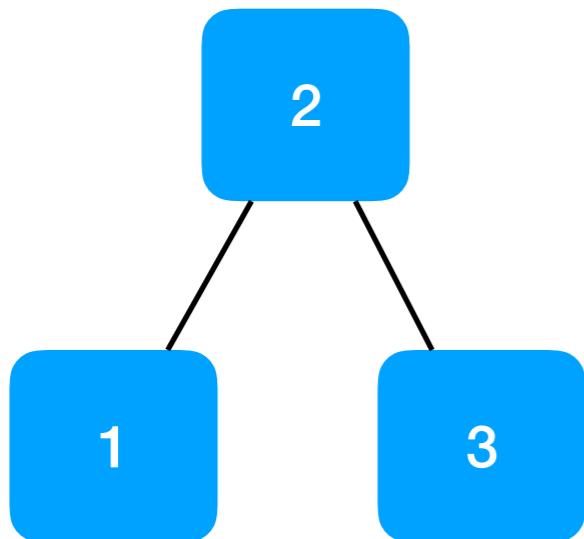
Vrai ou Faux ?

- Étant donné un parcours préfixe d'un BST contenant N clefs distinctes.
 - Est-il possible de reconstruire la forme du BST sur base du résultat du parcours?
 - Si oui, écrivez le pseudo-code d'un algorithme pour le faire, si non, donnez un contre-exemple qui justifie votre réponse.

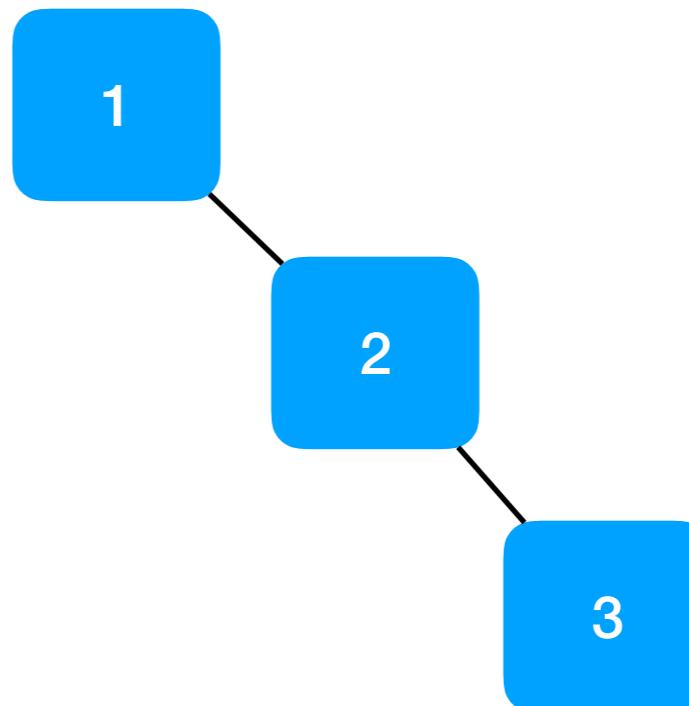
Revenons à notre contre exemple

- $\text{prefixe}(T_1) = 2, 1, 3$ et $\text{prefixe}(T_2) = 1-2-3$

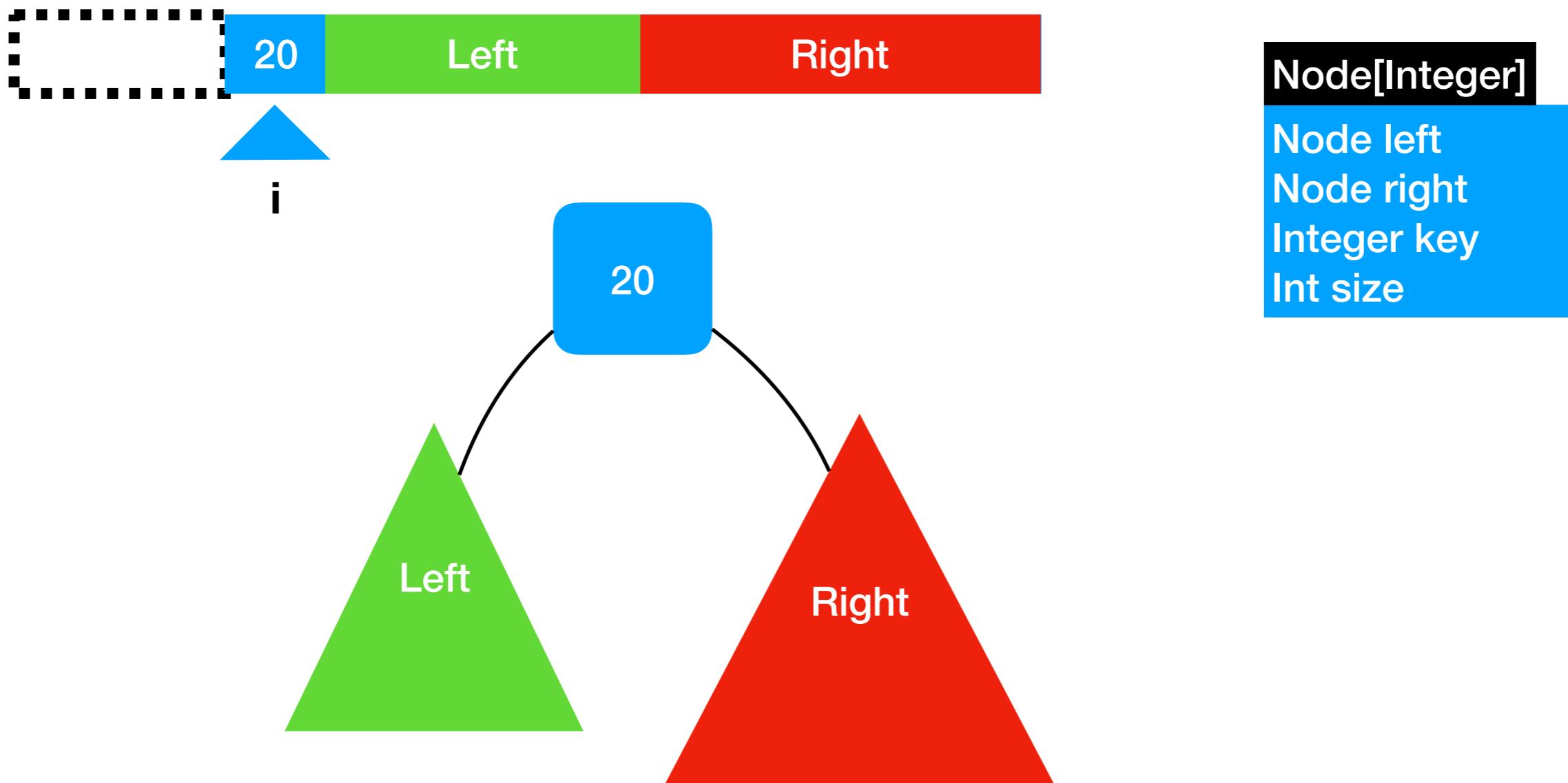
- T_1



- T_2



Idée de l'algorithme

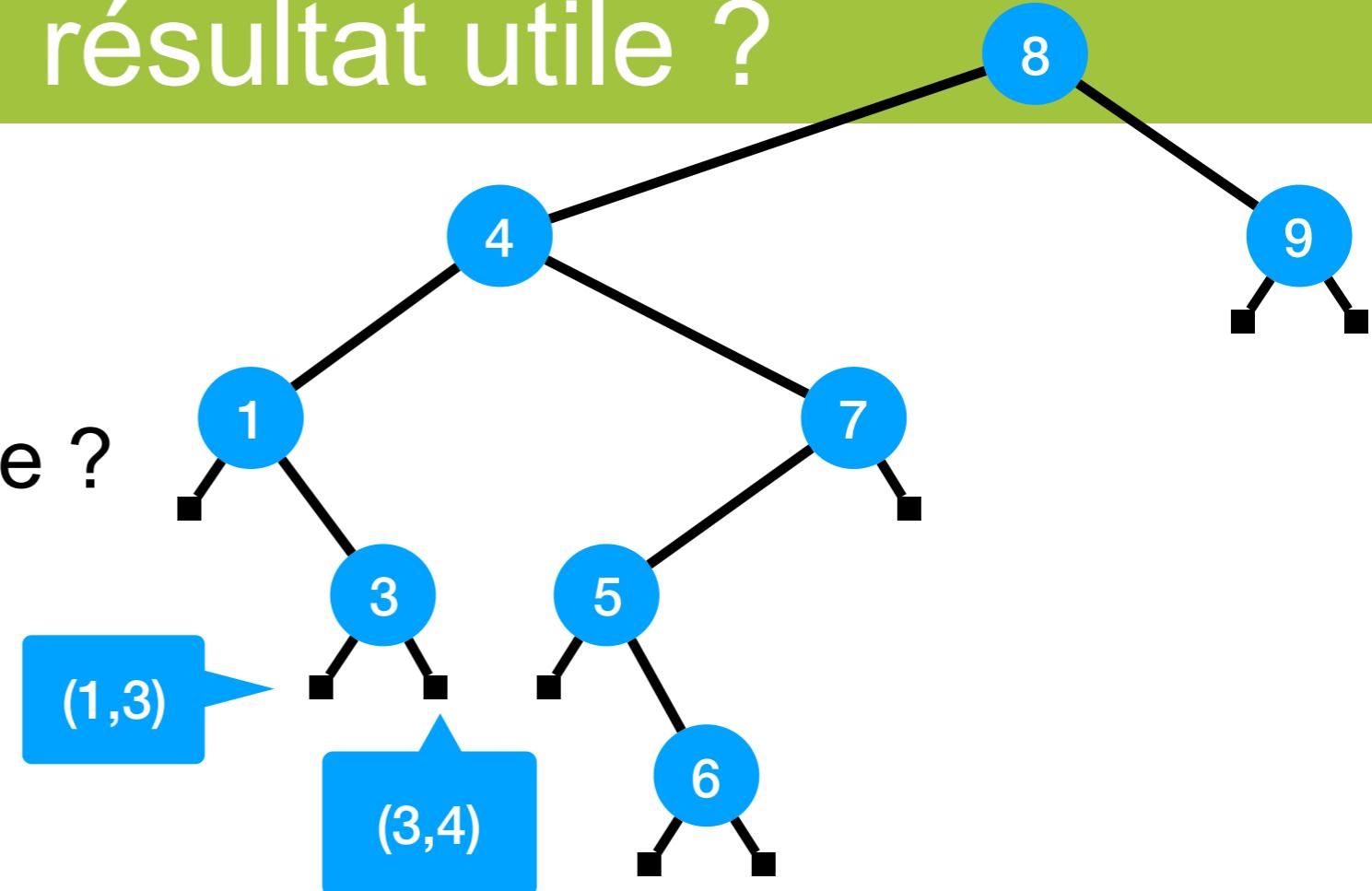


```
private Node readBSTFromPreorder(int [] input, int i, ...) {  
    Node left = readBSTFromPreorder(input,i+1,...);  
    Node right = readBSTFromPreorder(input,i+left.size+1,...);  
    return new Node(input[i],left,right,left.size+right.size+1);  
}
```

Pourquoi est-ce un résultat utile ?

- Et pour les noeuds feuille ?

8,4,1,3,7,5,6,9



- Lorsque $i = 3$, on va lancer récursivement sur au départ de $i=4$ ($\text{key}=7$) la lecture de l'arbre de gauche:
 - ▶ Pourquoi 7 ne peut-il pas être un enfant gauche de 7 sachant que le préfixe 8,4,1,3 a déjà été lu ?
 - ▶ Car on s'attend à une clef entre 1 et 3 sur la gauche, et entre 3 et 4 sur la droite

Algo: Recursive Procedure

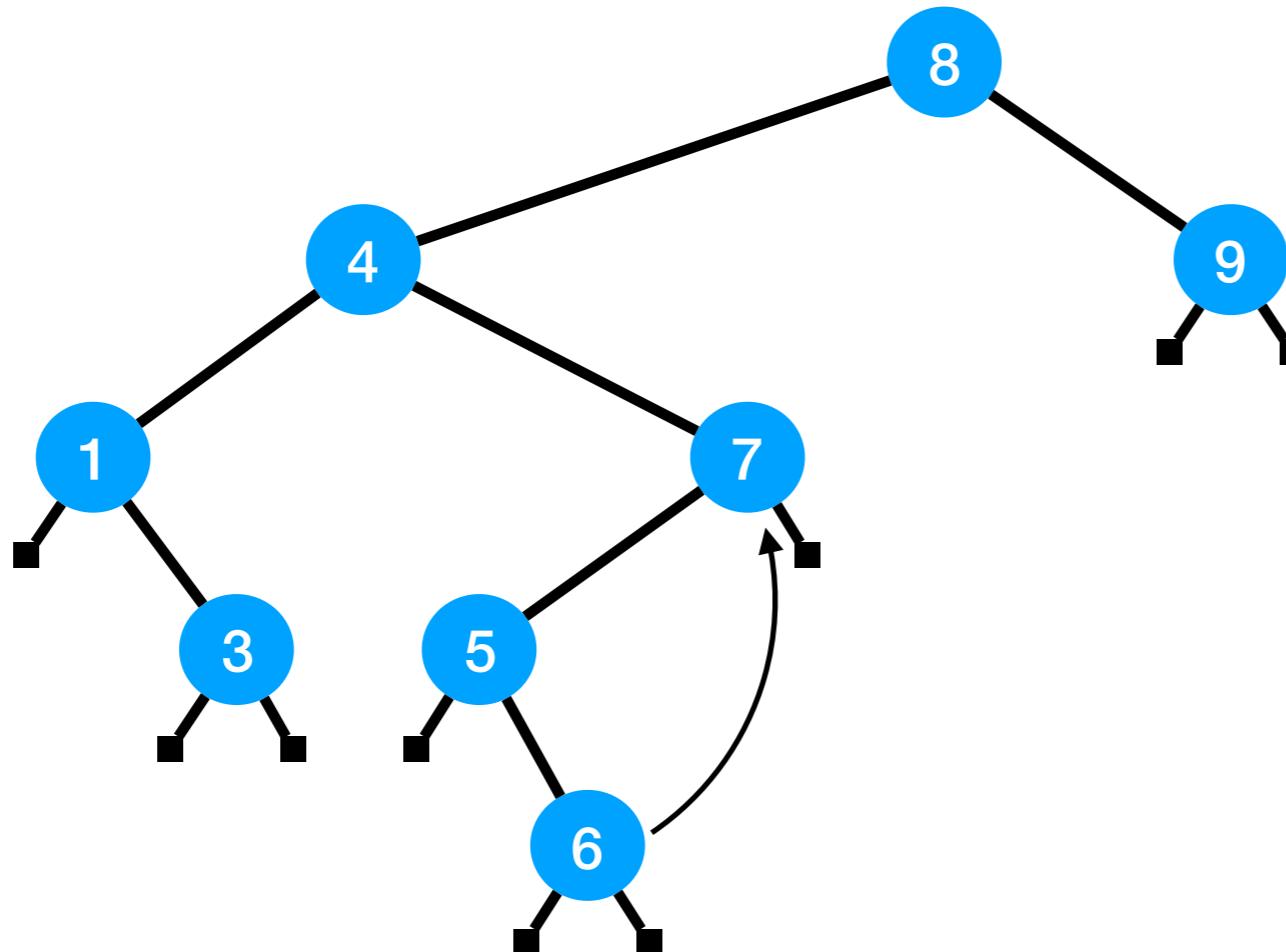
```
private static Node readBSTFromPreorder(int [] input) {  
    return readBSTFromPreorder(input, 0, Integer.MIN_VALUE, Integer.MAX_VALUE);  
}
```

Lecture du plus long BST possible enraciné au noeud `input[i]` ayant toutes ses clefs comprises entre `min` et `max`

`@param input` = les clefs d'un BST ordonnées selon un parcours préordre
`@param i` = une position dans `input`
`@param min` la valeur de clef minimum
`@param max` la valeur de clef maximum

```
private static Node readBSTFromPreorder(int [] input, int i, int min, int max) {  
    if (i < input.length && input[i] > min && input[i] < max) {  
        Node left = readBSTFromPreorder(input, i+1, min, input[i]);  
        Node right = readBSTFromPreorder(input, i+left.size+1, input[i], max);  
        return new Node(input[i], left, right, left.size+right.size+1);  
    } else {  
        return EMPTY;  
    }  
}
```

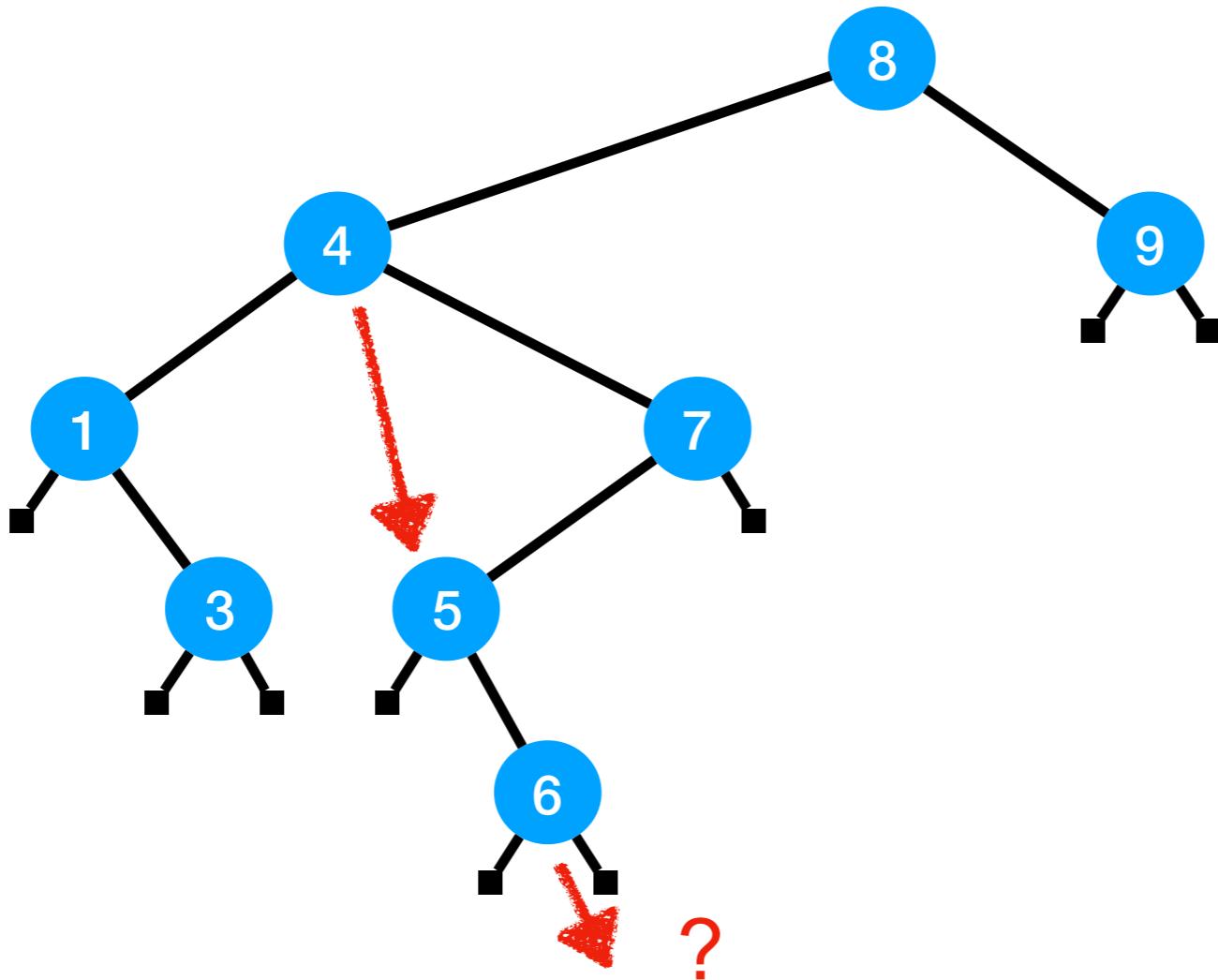
- Étant donné un arbre ordonné de N clefs distinctes et une clef x , est-il possible de trouver la plus petite clef strictement plus grande que x en temps logarithmique dans le pire cas ?



Hauteur BST

- La hauteur attendue d'un BST résultant de l'insertion de N clefs distinctes dans un ordre aléatoire dans un arbre initialement vide est en moyenne logarithmique.

- Soit x un noeud dans un BST.
- Le successeur de x (le noeud contenant la clef suivante dans l'ordre croissant) est le noeud le plus à gauche dans l'arbre de droite de x ?

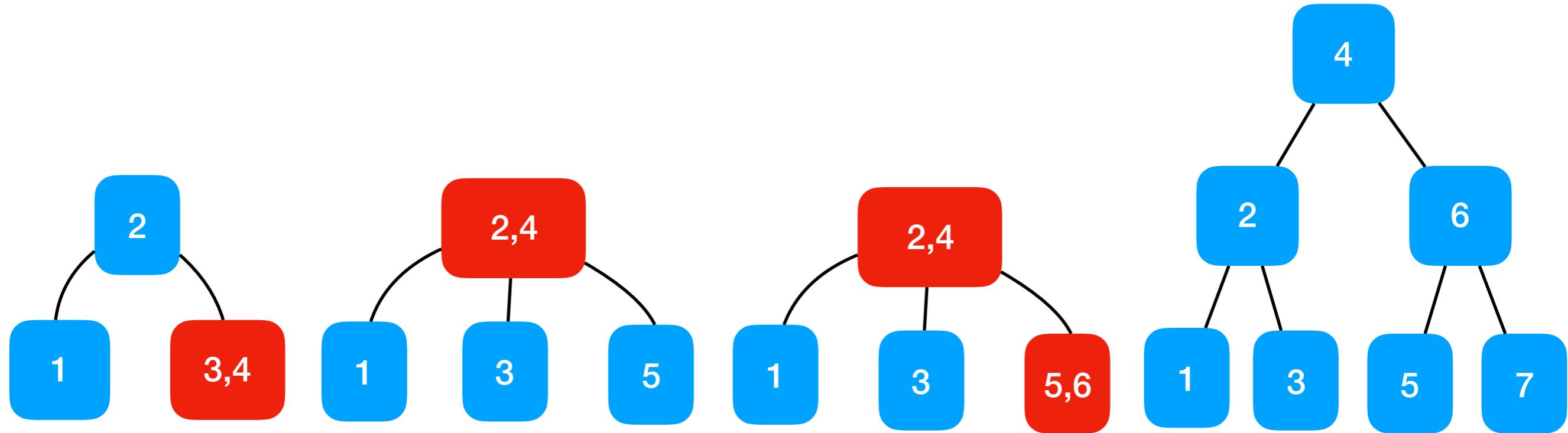


- La hauteur max d'un 2-3 tree avec N clefs est $\sim \log_3 N$
- La hauteur d'un 2-3 tree est entre
 - $\text{floor}(\log_3 N)$ (tous des 3 nodes) et
 - $\text{floor}(\log N)$ (tous des 2 nodes)
- C'est donc faux, $\sim \log_3 N$ est la hauteur min d'un 2-3 tree.

- Pour l'insertion de N clefs dans l'ordre croissant dans un red-black BST initialement vide. Le nombre de changements de couleur de la dernière insertion est au plus 3. Le nombre de changements s'entend comme la somme des différences en valeur absolue entre le nombre de rouges après insertion moins le nombre de rouges avant insertion.
- On traduit la question en arbre 2-3: Pour l'insertion de N clefs dans l'ordre croissant dans un arbre 2-3 initialement vide. Lors de la dernière insertion, j'ai supprimé au plus 3 3-nodes.

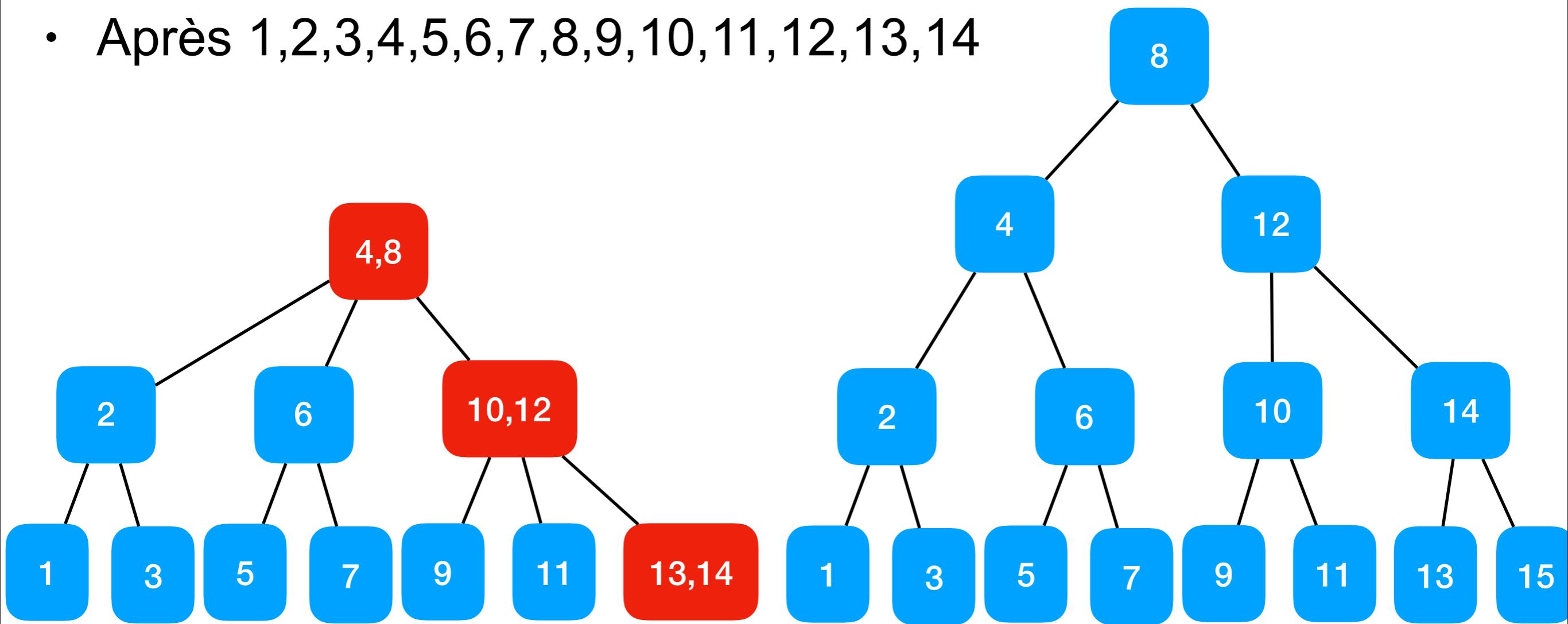


- Essayons de trouver un contre exemple:
- On insère 1,2,3,4,5,...

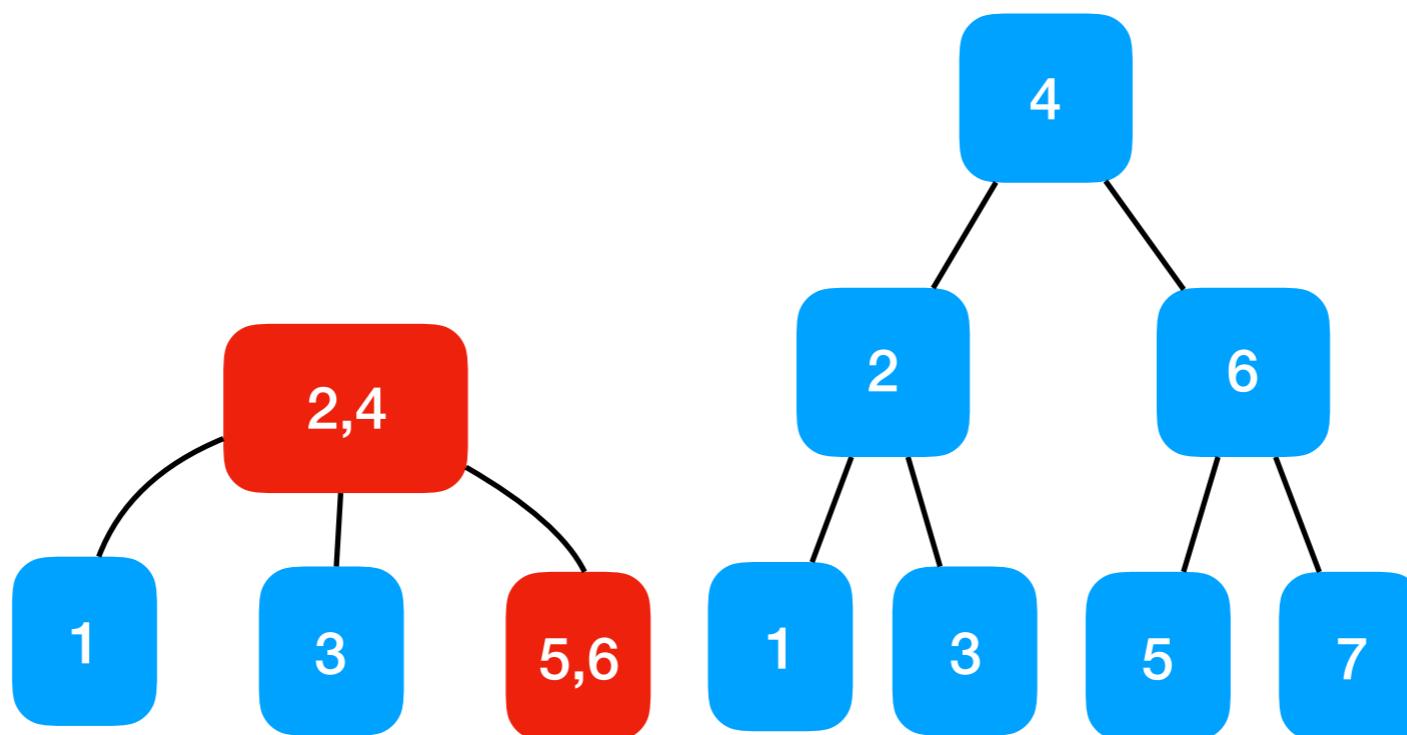


- Essayons de trouver un contre exemple:
- On insère 1,2,3,4,5,6,7 **cette dernière insertion supprime 2 noeuds 2-3.** A l'insertion de 15, le nombre de changements sera 3, car la branche de droite seront toutes saturées en 3 nodes et la “bulle” doit remonter jusqu'à la racine.

- Après 1,2,3,4,5,6,7,8,9,10,11,12,13,14



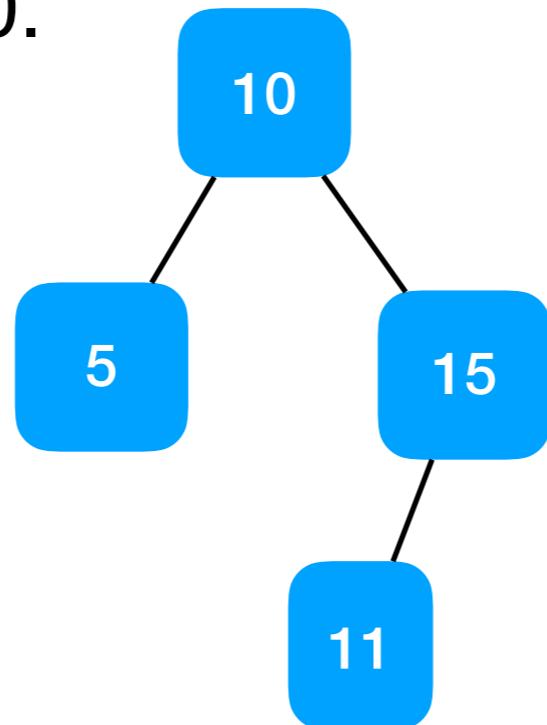
- Un red-black BST obtenu après insertion de $N > 1$ clefs dans un arbre initialement vide possède au moins un lien rouge.
- Traduction: Il y a au moins un 3-node dans un red-black ?



- Dans un red-black BST de N noeuds, la hauteur noire (i.e. le nombre de liens noirs de chaque chemin depuis la racine vers un lien null) est maximum $\log N$.
- Traduction: la hauteur dans un arbre 2-3 est maximum $\log N$.
 - Oui puisque la hauteur d'un 2-3 tree est entre
 - $\text{floor}(\log_3 N)$ (tous des 3 nodes) et
 - $\text{floor}(\log N)$ (tous des 2 nodes)

- Imaginez un algorithme de tri utilisant un BST. A quoi ressemblerait cet algorithme ?
- Quelle serait la complexité de votre algorithme si le BST est remplacé par un red-black BST ?
 - BST: $O(n^2)$ pour la construction du BST car l'insertion prend $O(n)$, ensuite $O(n)$ pour faire le parcours infixé => $O(n^2)$ en tout.
 - red-black: $O(n \log(n))$ pour la construction du red-black car l'insertion prend $O(\log(n))$. Ensuite $O(n)$ pour faire le parcours infixé => $O(n \log(n))$ en tout

- Est-ce que l'opération de suppression dans un BST est "commutative" ? C'est à dire que supprimer x et ensuite directement y d'un BST (tel qu'implémenté dans le livre) laisse l'arbre dans même état que si on avait d'abord supprimé y et puis y ?
- Donner un contre-exemple ou argumenter pourquoi c'est effectivement toujours le cas.
- Pour vous aider, considérez l'arbre suivant et les opérations de suppression de 5 et 10.

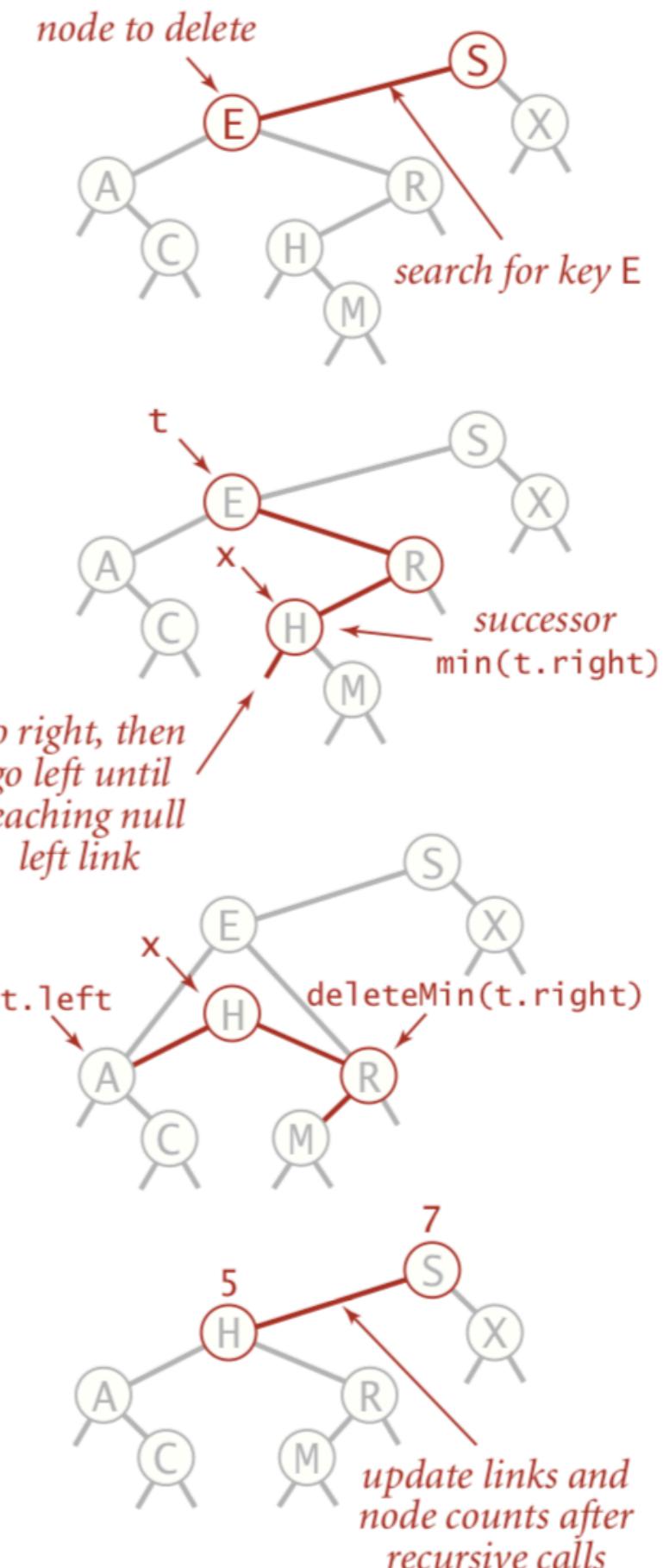


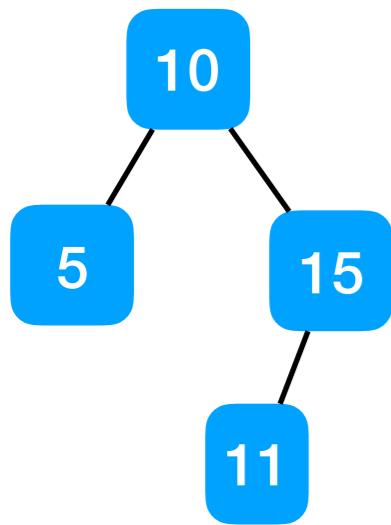
Rappel sur le delete

```
public void delete(Key key)
{   root = delete(root, key);  }

private Node delete(Node x, Key key)
{
    if (x == null) return null;
    int cmp = key.compareTo(x.key);
    if      (cmp < 0) x.left  = delete(x.left,  key);
    else if (cmp > 0) x.right = delete(x.right, key);
    else
    {
        if (x.right == null) return x.left;
        if (x.left == null) return x.right;
        Node t = x;
        x = min(t.right); // See page 407.
        x.right = deleteMin(t.right);
        x.left = t.left;
    }
    x.N = size(x.left) + size(x.right) + 1;
    return x;
}
```

Successeur





```

public void delete(Key key)
{   root = delete(root, key);  }

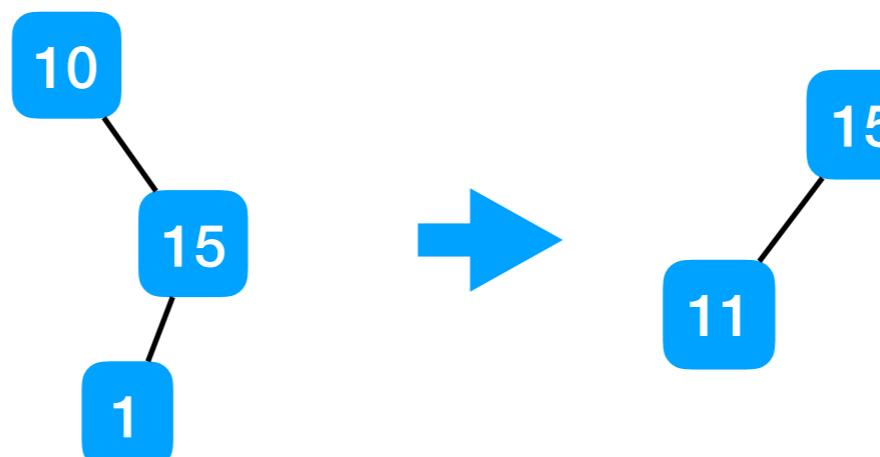
private Node delete(Node x, Key key)
{
    if (x == null) return null;
    int cmp = key.compareTo(x.key);
    if      (cmp < 0) x.left  = delete(x.left,  key);
    else if (cmp > 0) x.right = delete(x.right, key);
    else
    {
        if (x.right == null) return x.left;
        if (x.left == null) return x.right;
        Node t = x;
        x = min(t.right); // See page 407.
        x.right = deleteMin(t.right);
        x.left = t.left;
    }
    x.N = size(x.left) + size(x.right) + 1;
    return x;
}

```

- Delete 10 et puis 5

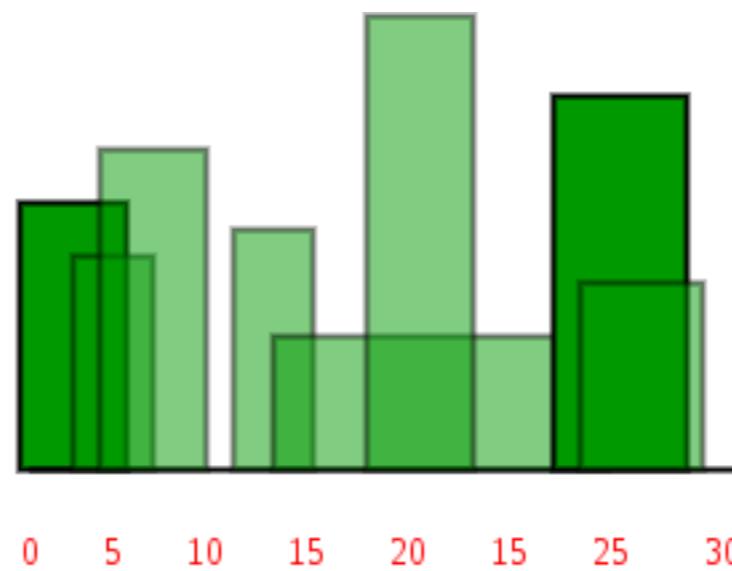


- Delete 5 et puis 10



Question Avancée: Skyline

des building (comme sur la figure), comment reconstruire le contour de la silhouette de tous les building sans chevauchement ?



Input: Array of buildings, every building is represented by triplet (left, ht, right)
{ (1,11,5), (2,6,7), (3,13,9), (12,7,16), (14,3,25), (19,18,22), (23,13,29), (24,4,28) }
Output: Skyline (an array of rectangular strips)
A strip has x coordinate of left side and height
(1, 11), (3, 13), (9, 0), (12, 7), (16, 3), (19, 18), (22, 3), (25, 0)