



LINFO 1121
DATA STRUCTURES AND ALGORITHMS



TP3: Arbres de recherche

Question 3.1.1 Sequential vs BINARY SEARCH

	Sequential	Binary
Insertion	$O(n)$	$O(n)$ $O(n/2)$ moyen
Recherche	$O(n)$	$O(\log n)$

n : nombre d'éléments dans l'array

Question 3.1.1 Sequential vs BINARY SEARCH

	Sequential	Binary
Insertion	$O(n)$	$O(n)$ $O(n/2)$ moyen
Recherche	$O(n)$	$O(\log n)$

seq =

bin =

P : Nombre de put

G : Nombre de get

Question 3.1.1 Sequential vs BINARY SEARCH

Coûts liés aux puts

$$\text{seq} = \left(\sum_{i=1}^P i \right)$$

$$\text{bin} = \left(\sum_{i=1}^P i \right)$$

	Sequential	Binary
Insertion	O(n)	O(n) O(n/2) moyen
Recherche	O(n)	O(log n)

P : Nombre de put

G : Nombre de get

Question 3.1.1 Sequential vs BINARY SEARCH

Coûts liés aux puts

+ coûts liés aux gets

$$\text{seq} = \left(\sum_{i=1}^P i \right) + GP$$

$$\text{bin} = \left(\sum_{i=1}^P i \right) + G \log P$$

	Sequential	Binary
Insertion	O(n)	O(n) O(n/2) moyen
Recherche	O(n)	O(log n)

P : Nombre de put

G : Nombre de get

Question 3.1.1 Sequential vs BINARY SEARCH

$$\text{seq} = \left(\sum_{i=1}^P i \right) + GP$$

$$\text{bin} = \left(\sum_{i=1}^P i \right) + G \log P$$

$$GP > G \log P$$

$$\text{seq} > \text{bin}$$

3.1.2 Interpolation Search



Cherchons la valeur 51 !

Avec une **Binary Search** classique, on testerait la médiane

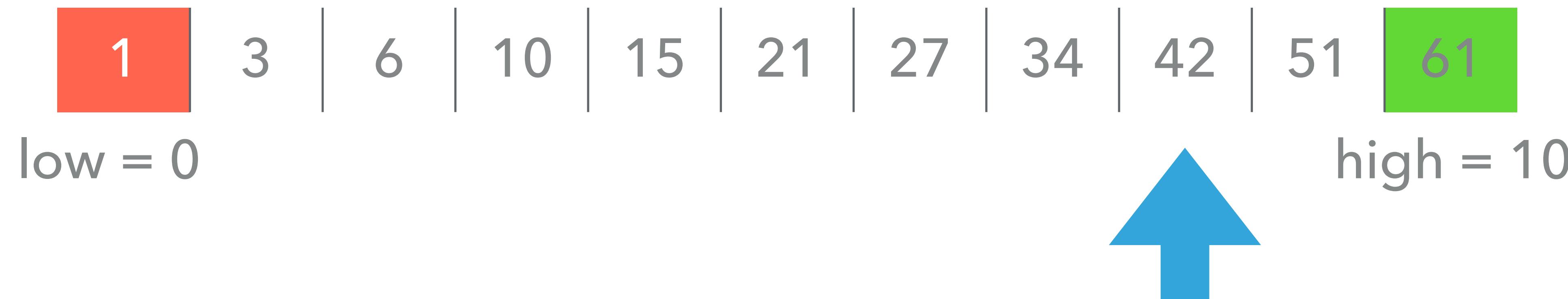
Ici, on regarde à l'index

$\text{low} + [(\text{high} - \text{low}) \cdot (\text{value} - \text{keys}[\text{low}]) / (\text{keys}[\text{high}] - \text{keys}[\text{low}])]$

3.1.2 Interpolation Search



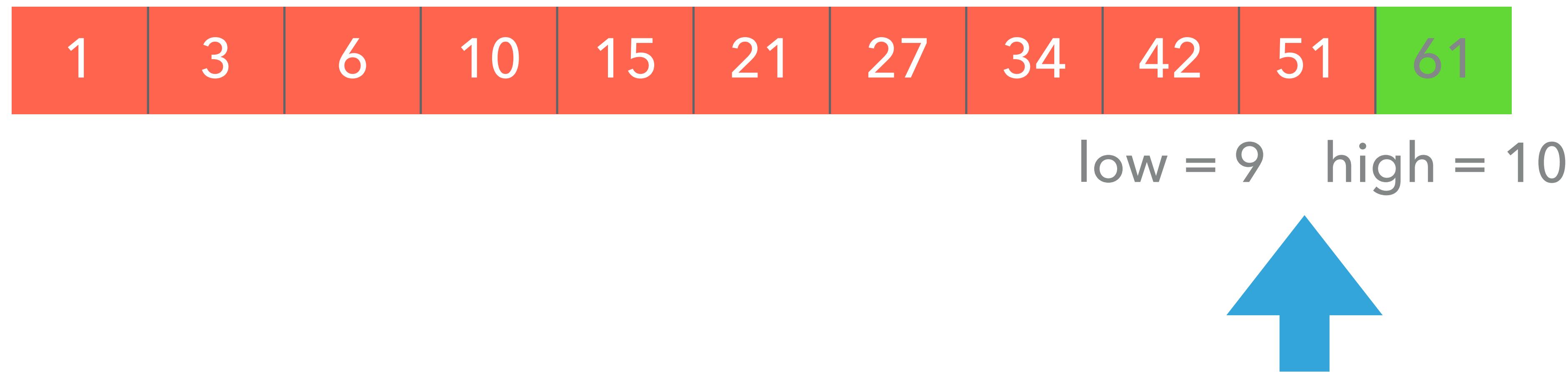
3.1.2 Interpolation Search



$$\text{index} = 0 + \lfloor (10 - 0) \cdot (51 - 1) / (61 - 1) \rfloor = 8$$

$$51 > 42 \rightarrow \text{lo} = \text{index} + 1$$

3.1.2 Interpolation Search



$$\text{index} = 9 + \lfloor (10 - 9) \cdot (51 - 51)/(61 - 1) \rfloor = 9$$

We found 51 !

3.1.2 Interpolation Search

```
public int rank(Key key) {  
    int low = 0  
    int high = keys.length - 1;  
    while (low <= high) {  
        int index = low + floor((high - low)(key - keys[lo])/( keys[high] - keys[low]));  
        if      (keys[index] < key) high = index - 1;  
        else if (keys[index] > key) low = index + 1;  
        else return mid;  
    }  
    return lo;  
}
```

Rappels sur les BSTs

Propriété d'un Binary Search Tree

C'est un arbre :

- C'est un graphe
- Connexe
- Possédant $n-1$ arêtes pour n noeuds
- Pas de cycles
- Au plus un chemin entre toute paire de noeuds

Il est binaire :

- Deux enfants par noeuds maximum
- Les noeuds en dessous, et à **gauche** doivent avoir des clés **plus petites**
- Les noeuds en dessous, et à **droite** doivent avoir des clés **plus grandes**

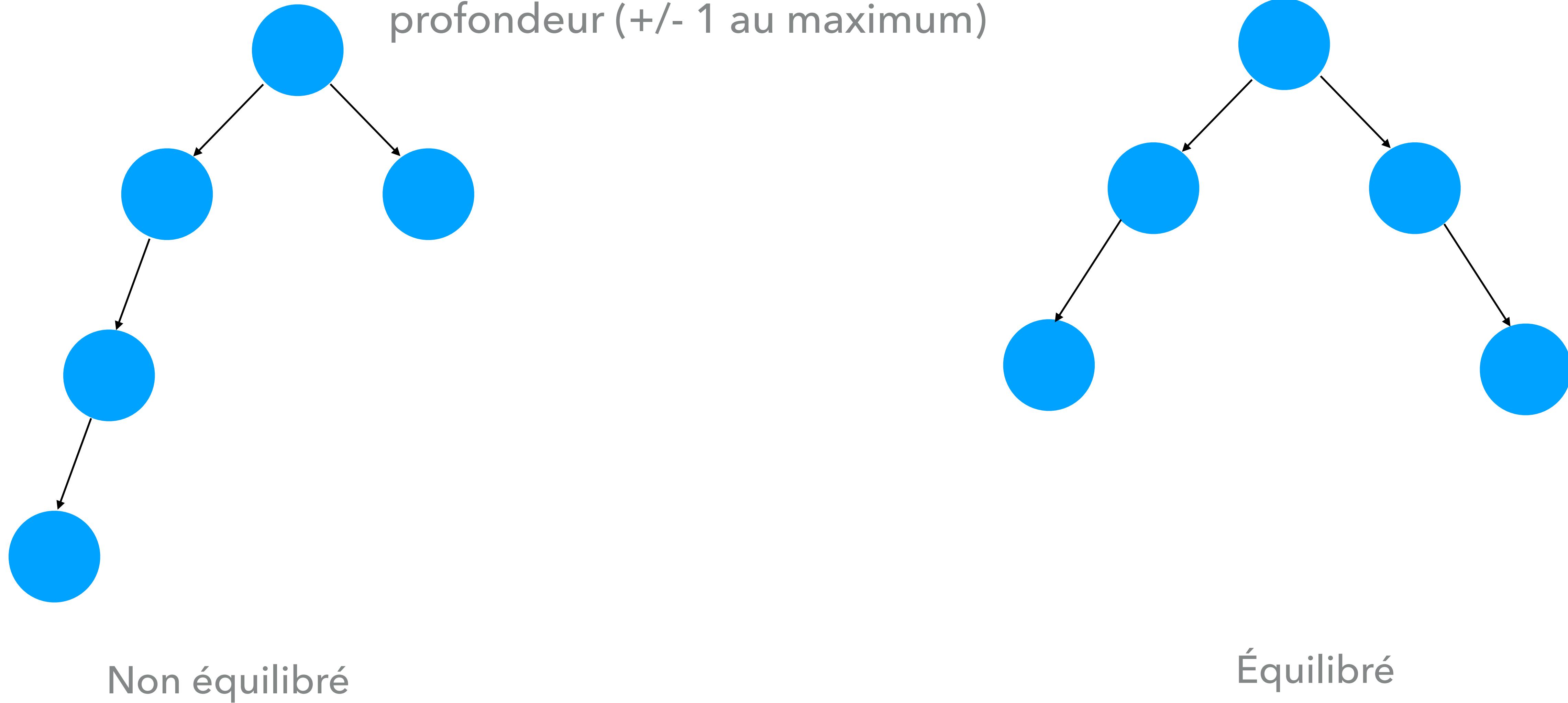
3.1.3 Caching key

```
public class BinarySearchST <Key extends Comparable<Key>, Value> {  
    private Key[] keys;  
    private Value[] vals;  
    private int N;  
    private Key cacheKey;  
    private int cacheRank;
```

```
    public int rank(Key key) {  
        if (key.compareTo(cacheKey) == 0)  
            return cacheRank;  
  
        int lo = 0, hi = N-1;  
        while (lo <= hi) {  
            int mid = lo + (hi - lo) / 2;  
            int cmp = key.compareTo(keys[mid]);  
            if (cmp < 0) hi = mid - 1;  
            else if (cmp > 0) lo = mid + 1;  
            else {  
                cacheKey = key;  
                cacheRank = mid;  
                return mid;  
            }  
        }  
        return lo;  
    }
```

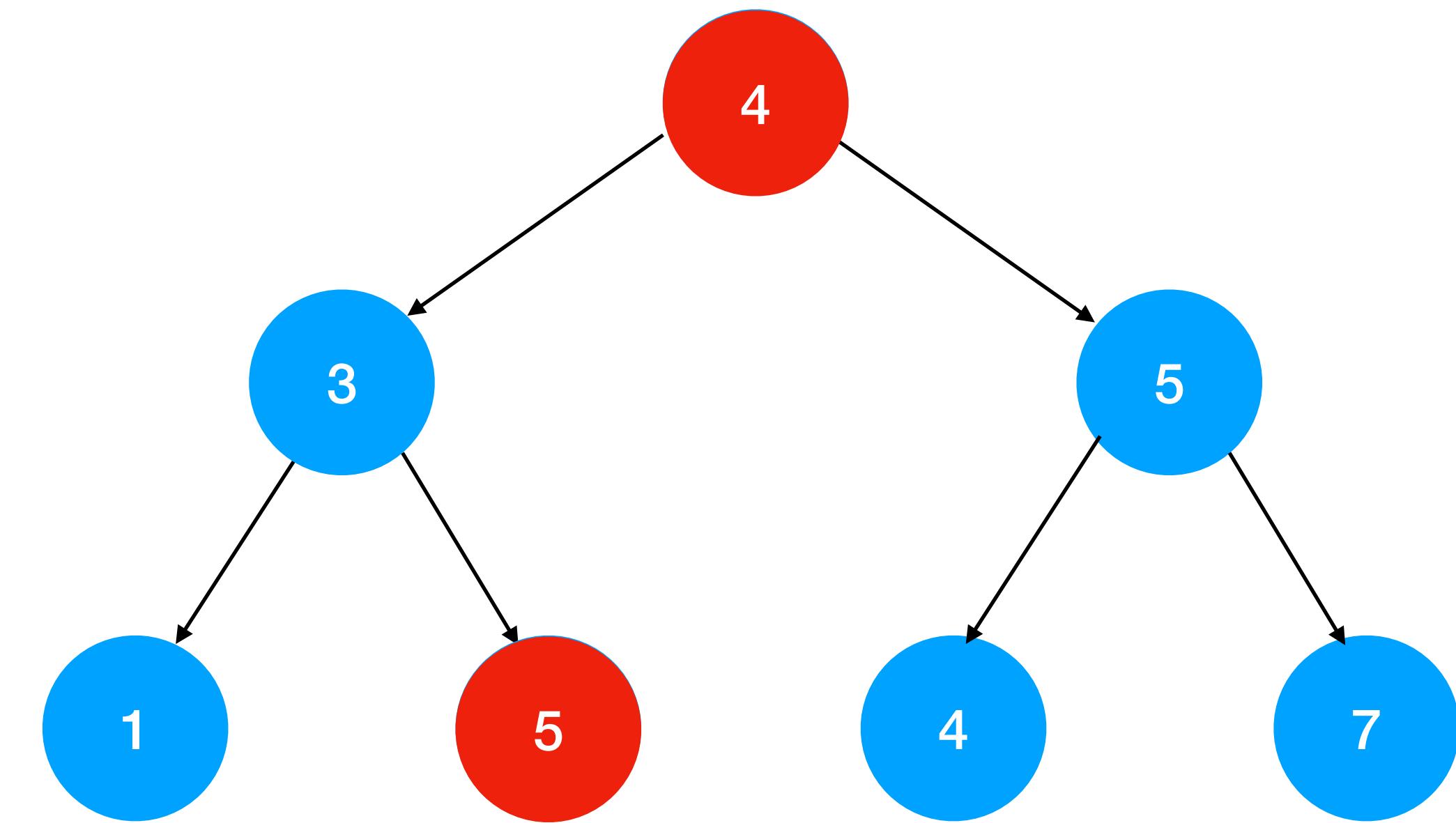
Équilibre d'un arbre

Équilibré : toutes les feuilles sont à la même profondeur (+/- 1 au maximum)



Question 3.1.4 isBST()

```
private boolean isBST(Node x) {  
    if (x == null) return true;  
    // Est-ce que l'enfant gauche contient bien une plus petite clé ?  
    if (x.key.compareTo(x.left.key) >= 0) return false;  
    // Est-ce que l'enfant droit contient bien une plus grande clé ?  
    if (x.key.compareTo(x.right.key) <= 0) return false;  
    return isBST(x.left) && isBST(x.right);  
}
```



Que retourne l'algorithme ?

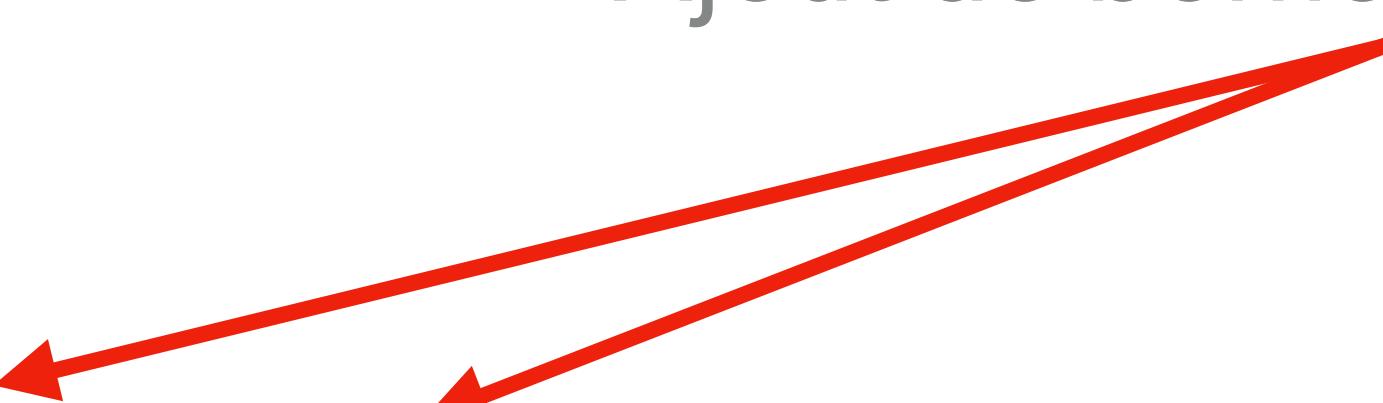
Est-ce un BST ?

Question 3.1.4 isBST()

```
private boolean isBST() {  
    return isBST(root, null, null);  
}
```

```
private boolean isBST(Node x, Key min, Key max) {  
    if (x == null) return true;  
    if (min != null && x.key.compareTo(min) <= 0) return false;  
    if (max != null && x.key.compareTo(max) >= 0) return false;  
    return isBST(x.left, min, x.key) && isBST(x.right, x.key, max);  
}
```

Ajout de bornes supérieures et inférieures

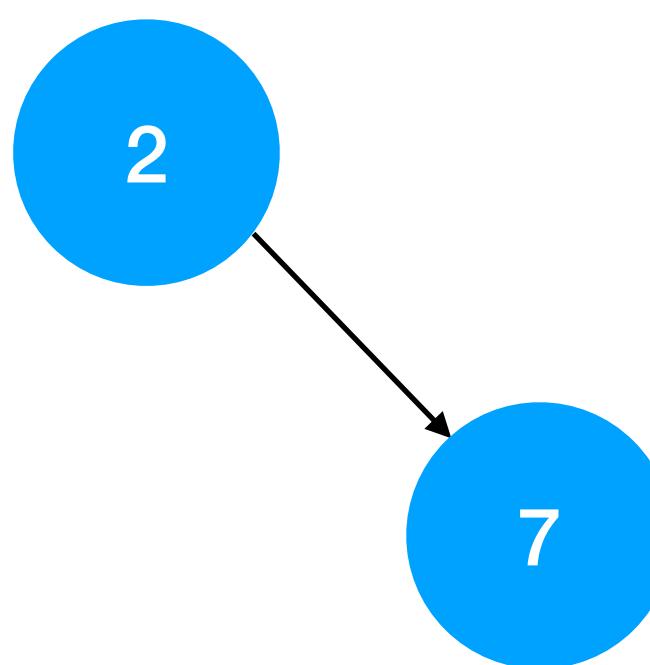


Question 3.1.5 VISITE POSSIBLES EN DFS

- 10,9,8,7,6,5
- 4,10,8,6,5
- 1,10,2,9,3,8,4,7,6,5
- 2,7,3,8,4,5
- 1,2,10,4,8,5

Question 3.1.5 VISITE POSSIBLES EN DFS

2,7,3,8,4,5

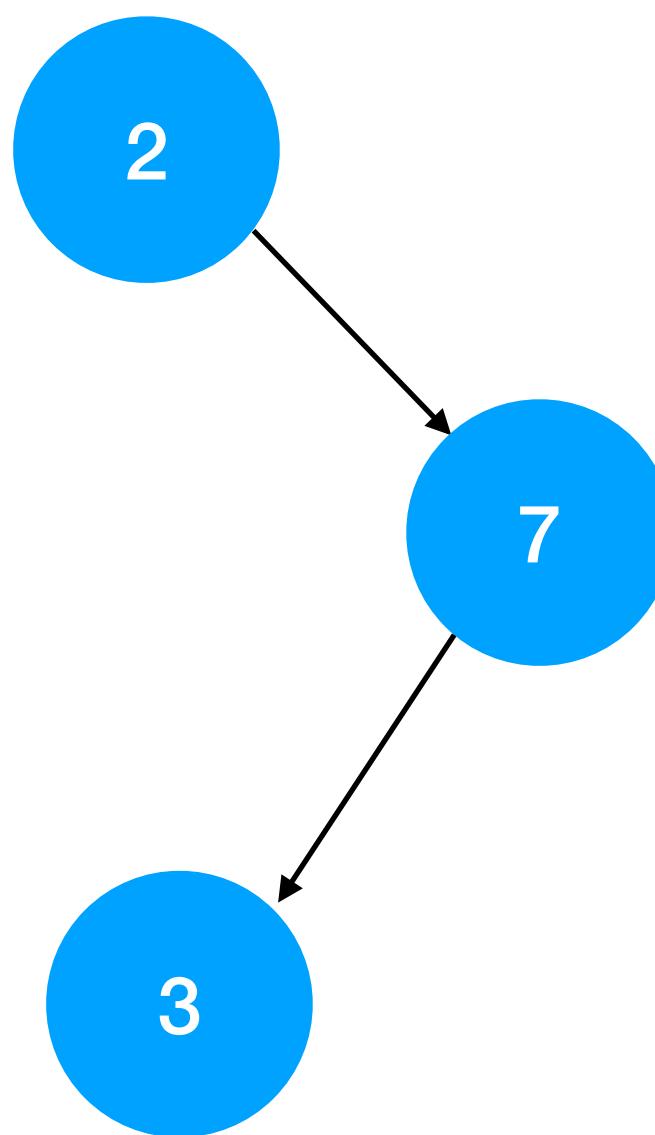


$$\begin{aligned} LB &= -\infty \\ UB &= +\infty \end{aligned}$$

$$7 > 2 \rightarrow LB = 2$$

Question 3.1.5 VISITE POSSIBLES EN DFS

2,7,3,8,4,5



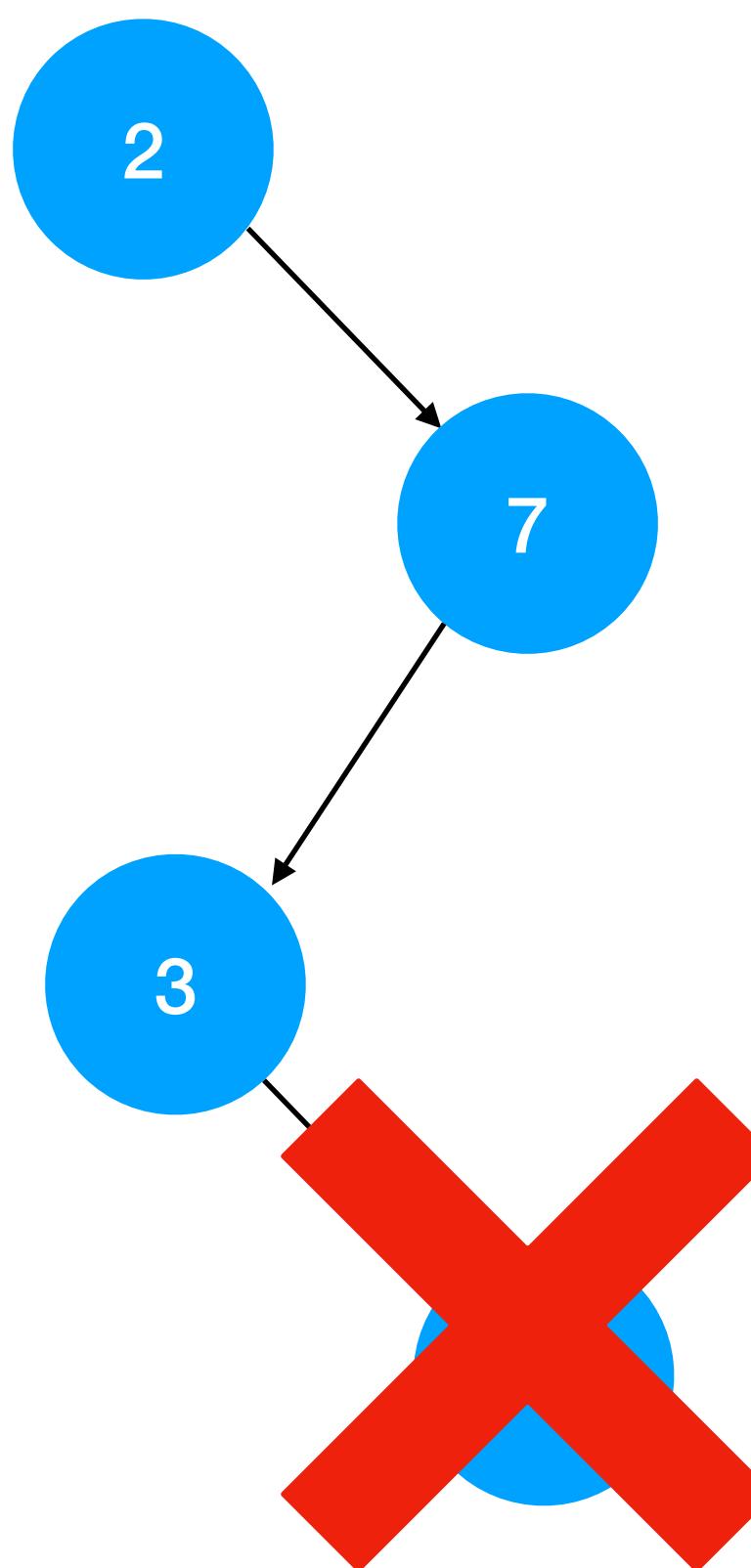
$$LB = 2$$

$$UB = +\infty$$

$$3 < 7 \rightarrow UB = 7$$

Question 3.1.5 VISITE POSSIBLES EN DFS

2,7,3,8,4,5



$$\begin{aligned} LB &= 2 \\ UB &= 7 \end{aligned}$$

$$8 > 7 \rightarrow 8 > UB$$

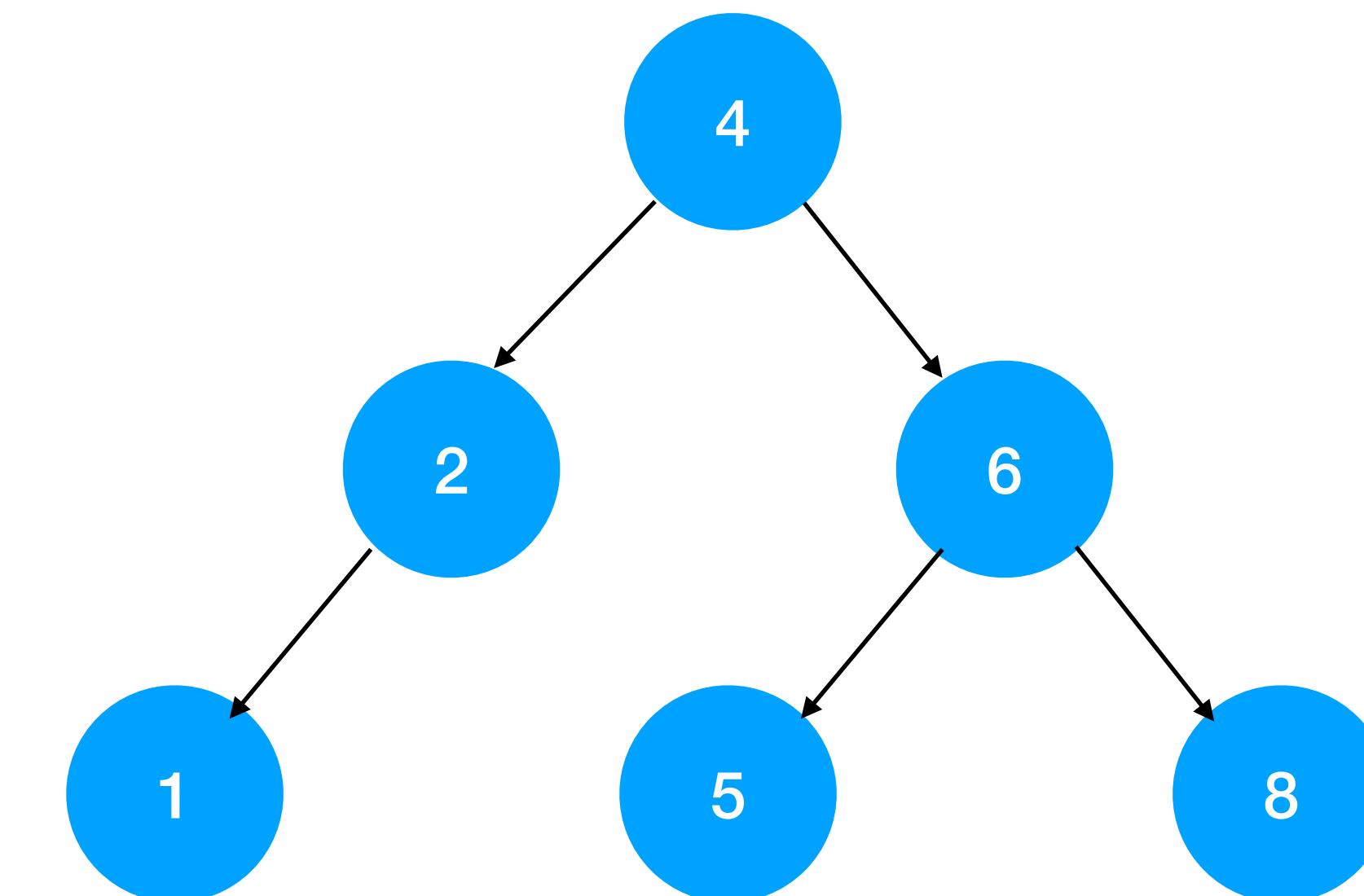
Question 3.1.5 VISITE POSSIBLES EN DFS

- 10,9,8,7,6,5 ✓
- 4,10,8,6,5 ✓
- 1,10,2,9,3,8,4,7,6,5 ✓
- 2,7,3,8,4,5 ✗
- 1,2,10,4,8,5 ✓

Question 3.1.7 Énumérer en ordre croissant les clés

Parcours infixé

```
public static void enumerate(Node parent) {  
    enumerate(parent.leftChild)  
    System.out.println(parent.key)  
    enumerate(parent.rightChild)  
}
```



Complexité : $O(\text{nombre de noeuds})$

Question 3.1.8 insertions

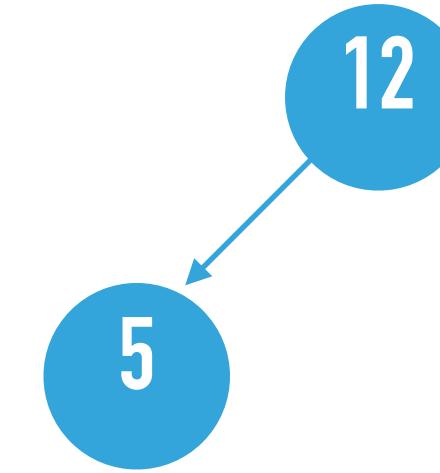
12, 5, 10, 3, 13, 14, 15, 17, 18, 15

Question 3.1.8 insertions

12

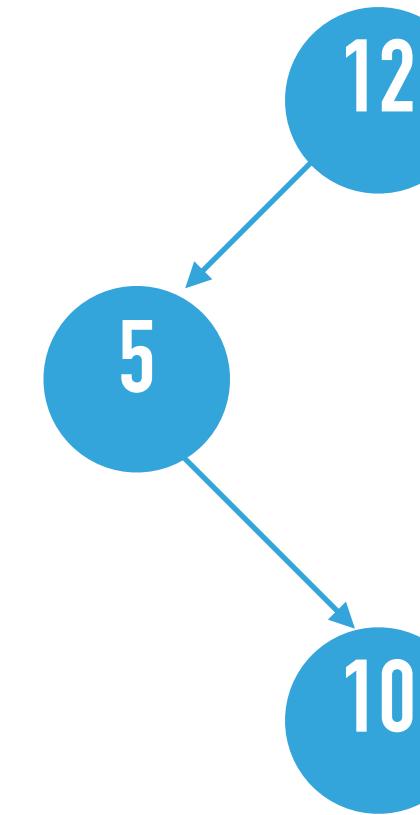
42, 5, 10, 3, 13, 14, 15, 17, 18, 15

Question 3.1.8 insertions



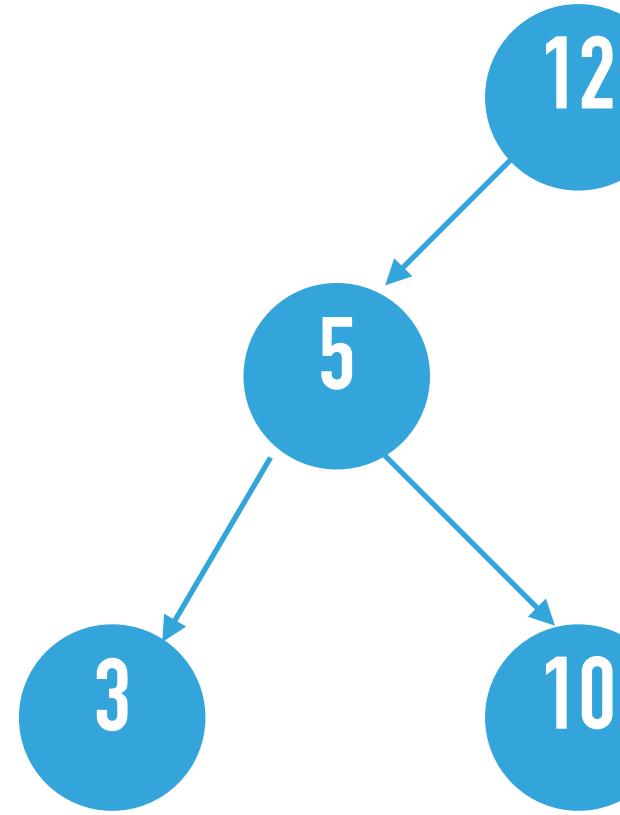
12, 5, 10, 3, 13, 14, 15, 17, 18, 15

Question 3.1.8 insertions



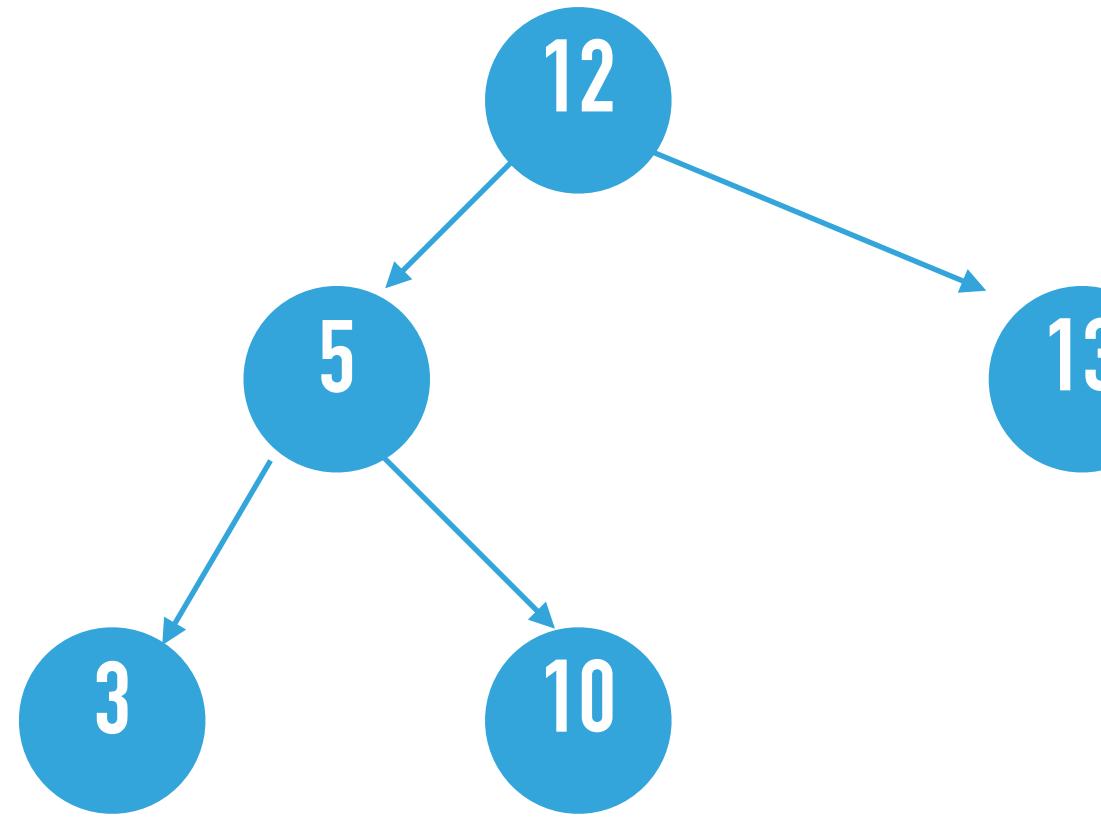
~~12, 5, 10, 3, 13, 14, 15, 17, 18, 15~~

Question 3.1.8 insertions



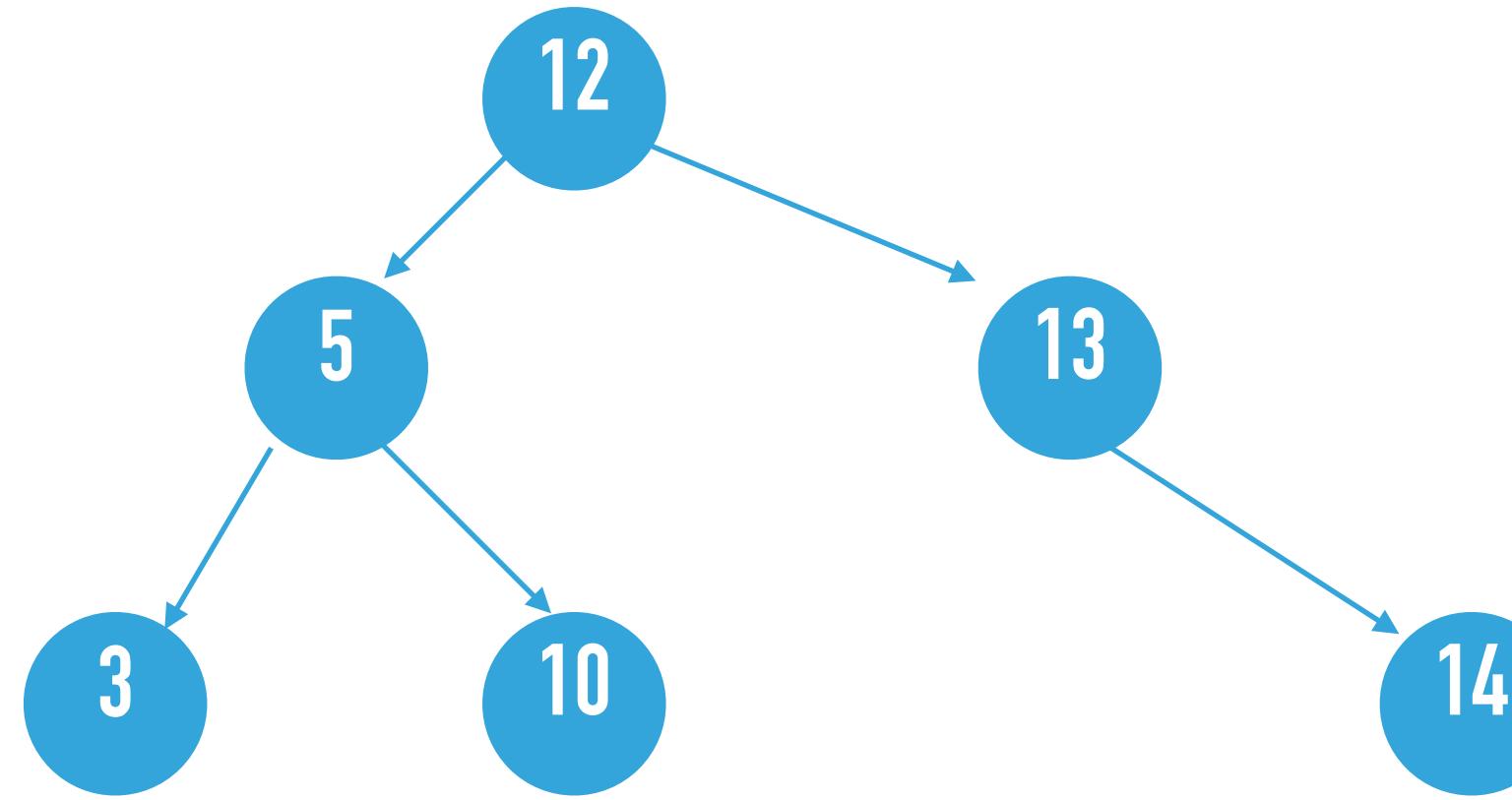
~~12, 5, 10, 3, 13, 14, 15, 17, 18, 15~~

Question 3.1.8 insertions



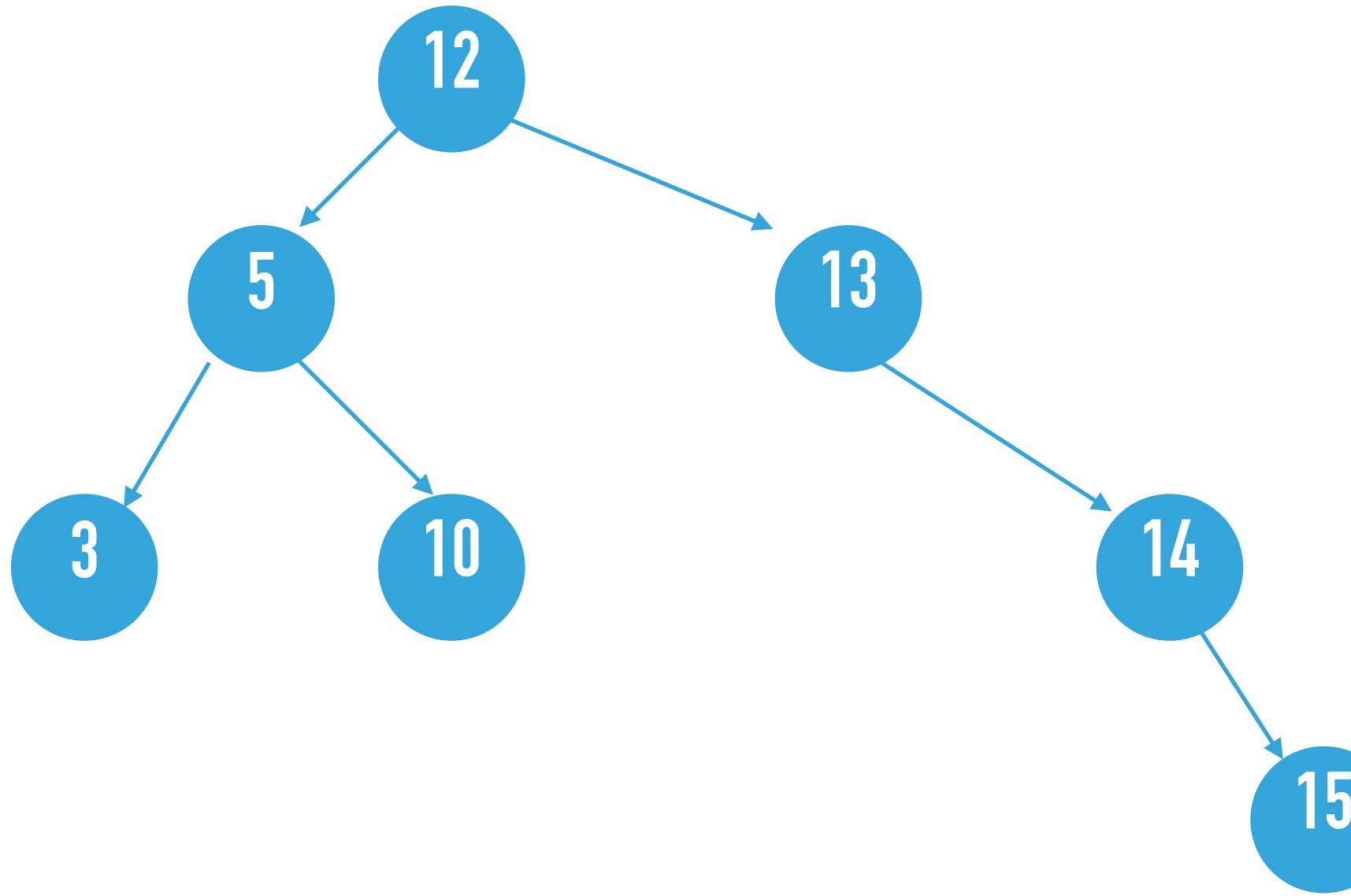
~~12, 5, 10, 3, 13, 14, 15, 17, 18, 15~~

Question 3.1.8 insertions



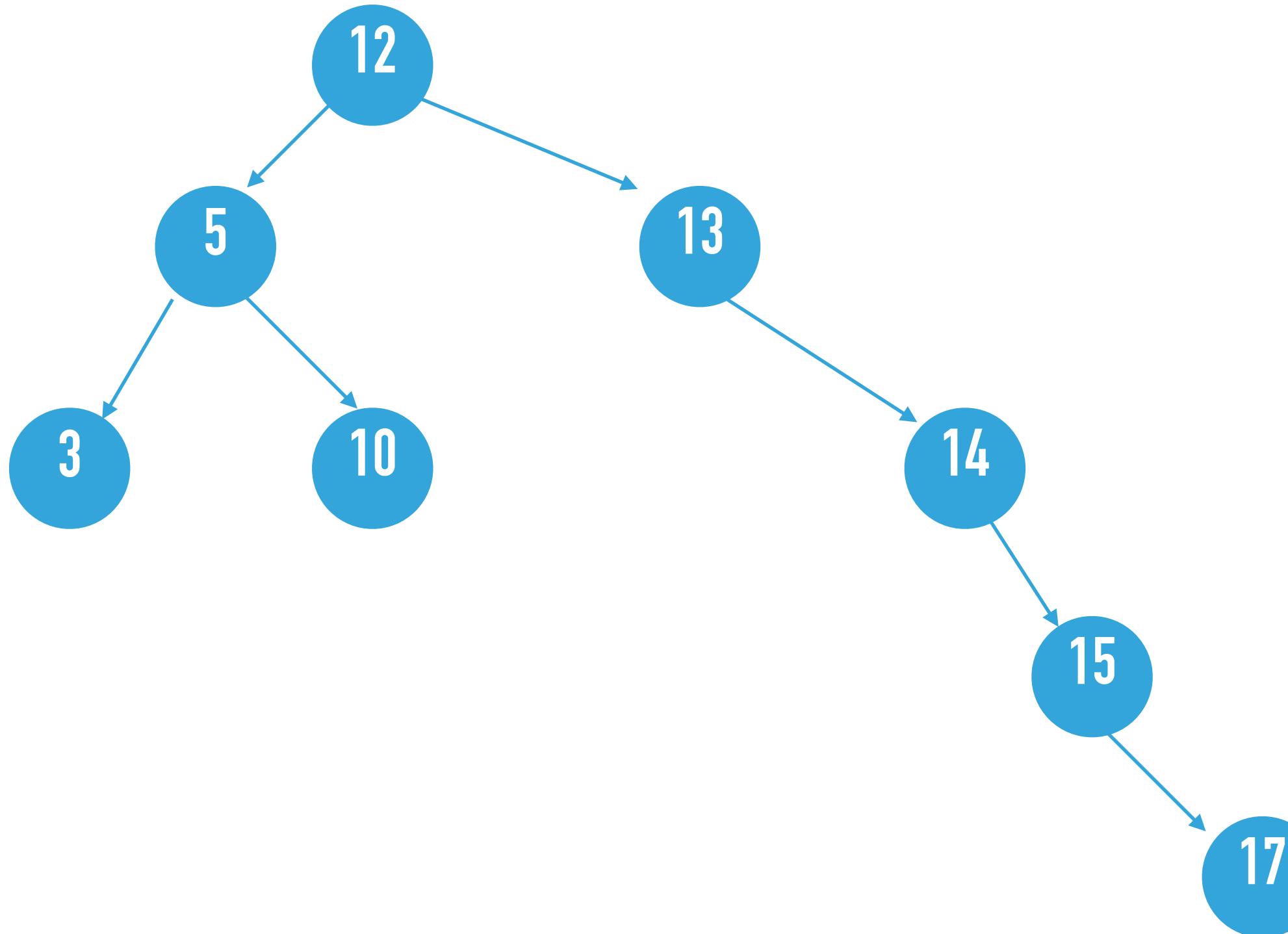
~~12, 5, 10, 3, 13, 14, 15, 17, 18, 15~~

Question 3.1.8 insertions



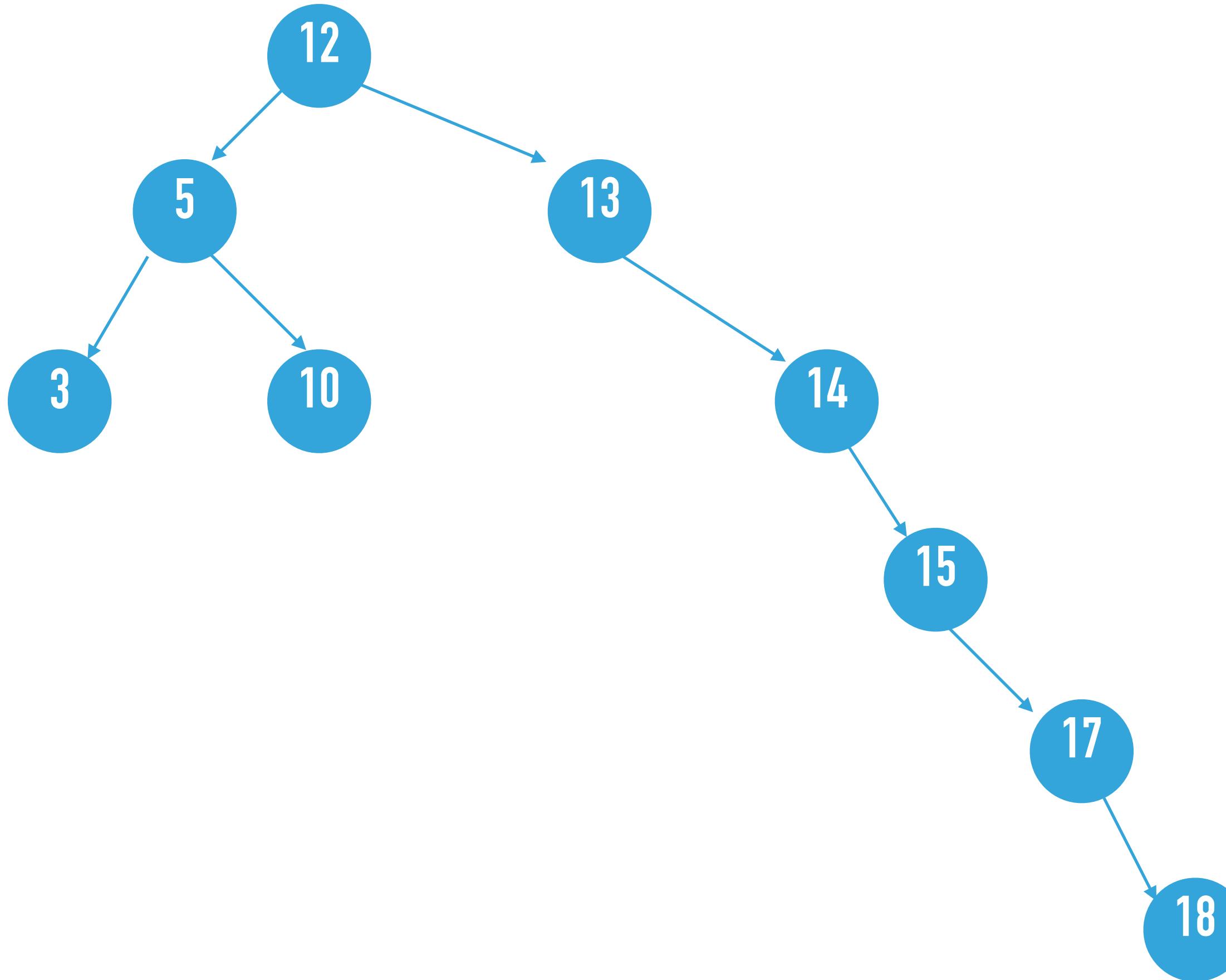
12, 5, 10, 3, 13, 14, 15, 17, 18, 15

Question 3.1.8 insertions



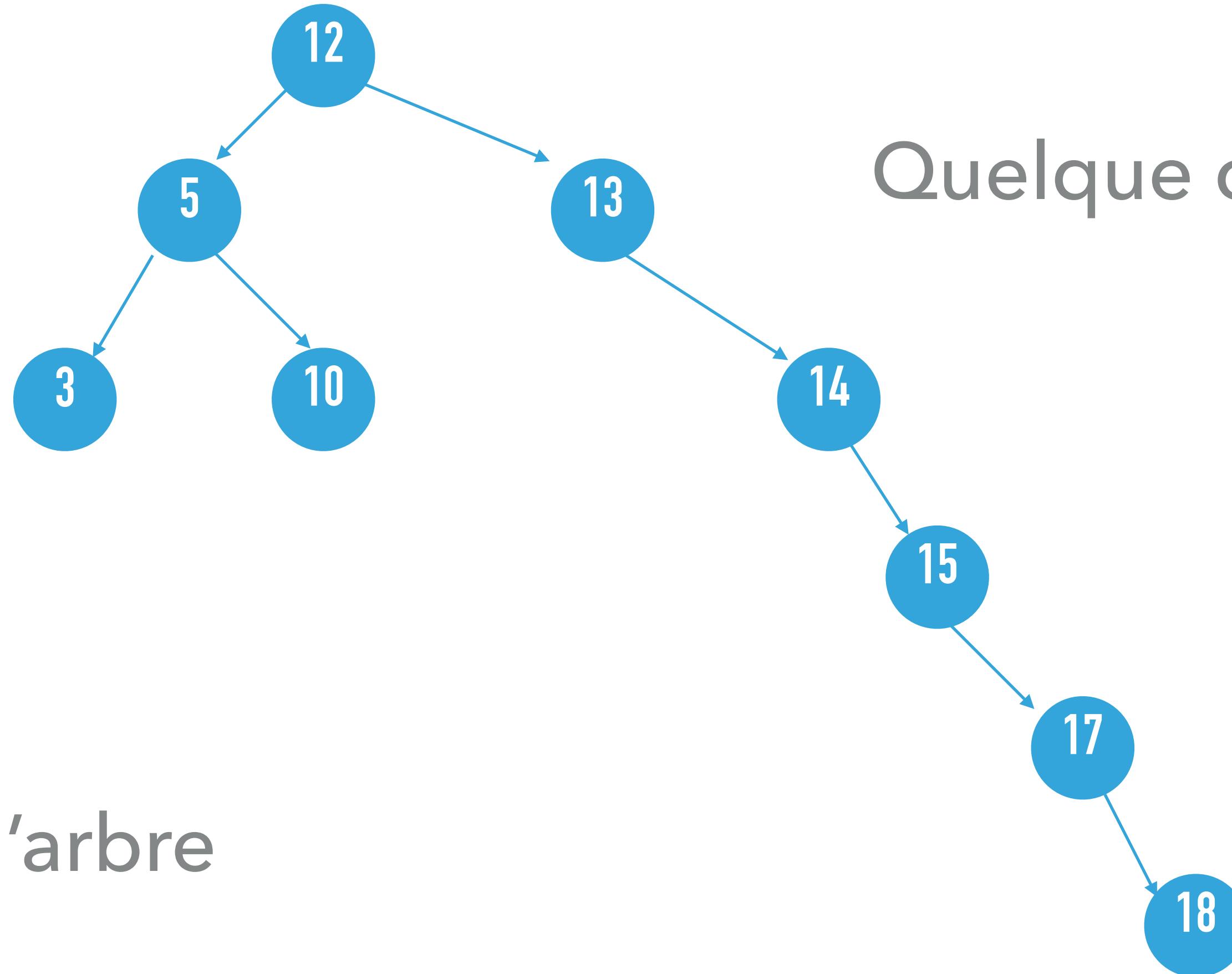
12, 5, 10, 3, 13, 14, 15, 17, 18, 15

Question 3.1.8 insertions



12, 5, 10, 3, 13, 14, 15, 17, 18, 15

Question 3.1.8 insertions



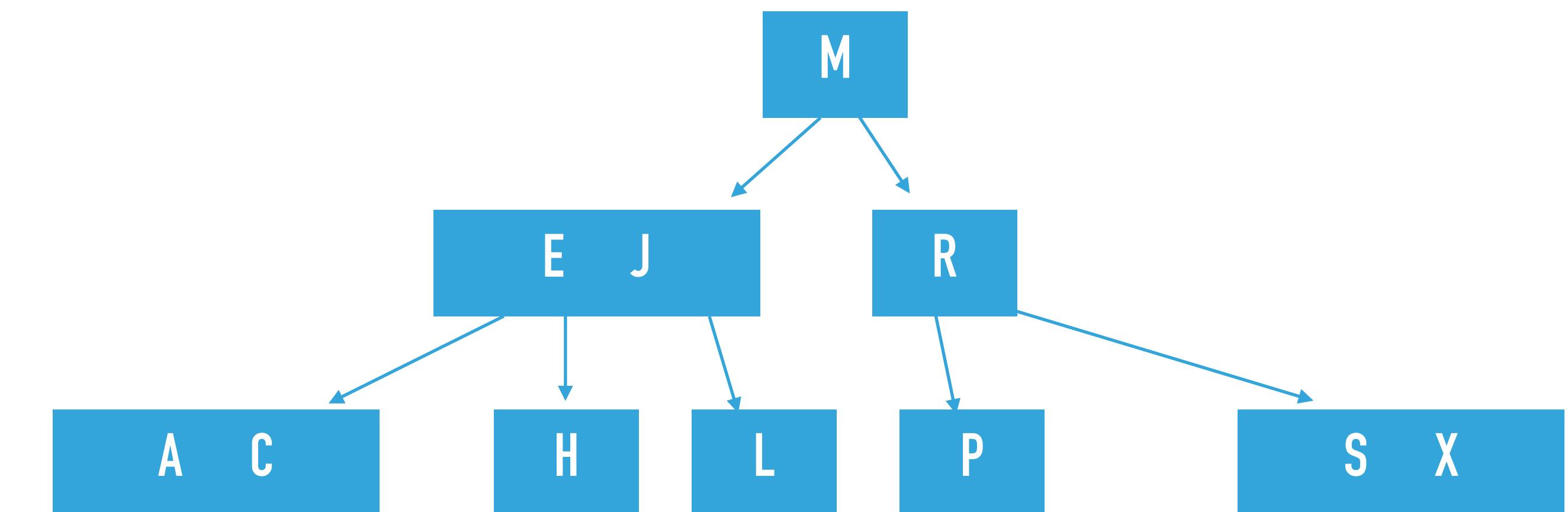
Quelque chose vous choque ?

15 est déjà dans l'arbre

12, 5, 10, 3, 13, 14, 15, 17, 18, 15

Arbres 2-3

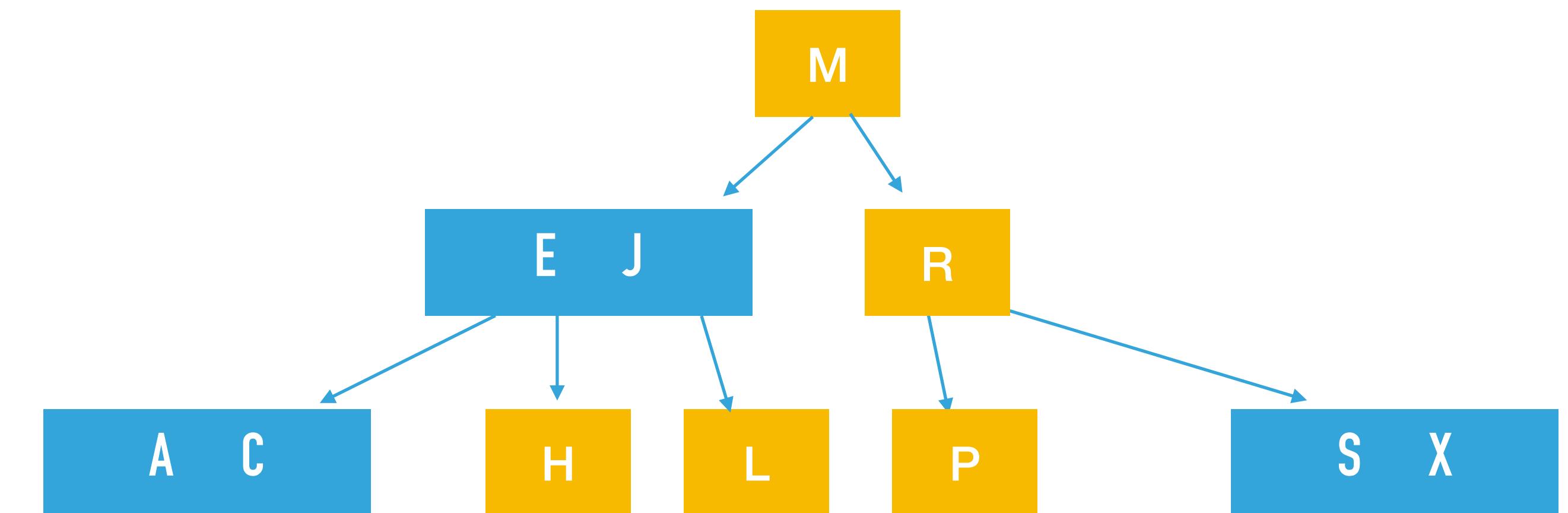
Arbre 2-3 contient soit :



Arbres 2-3

Arbre 2-3 contient soit :

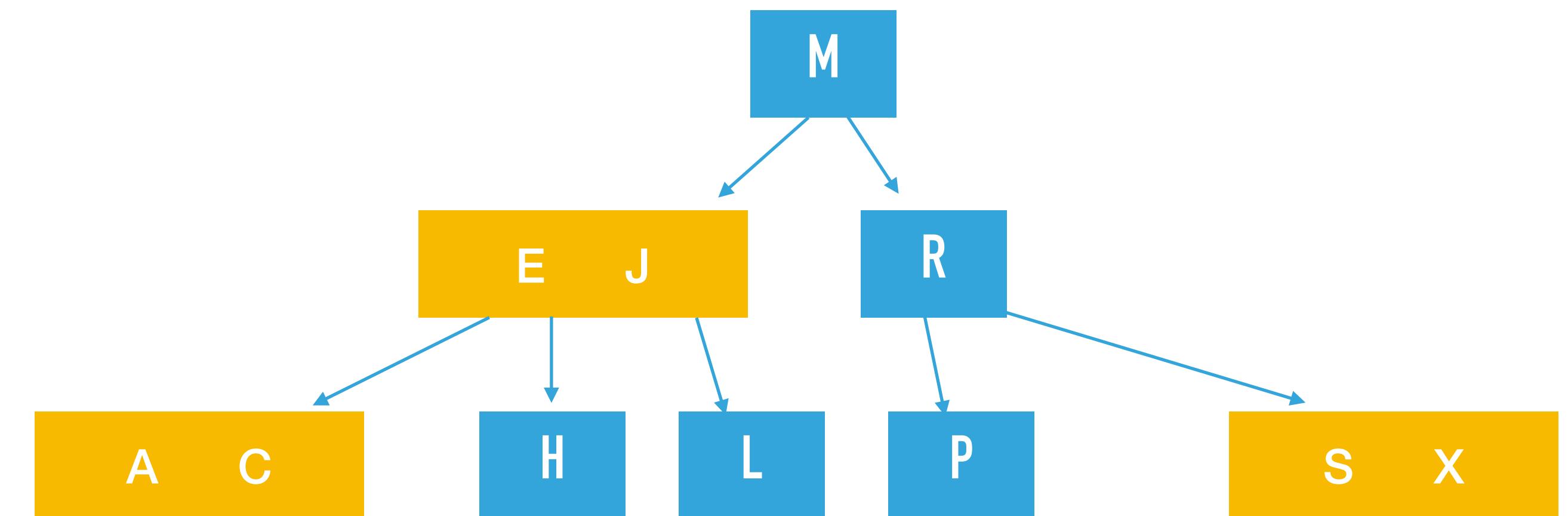
- 2-noeud, avec 1 clé et deux liens
- À gauche : clés plus petites
- À droite : clés plus grandes



Arbres 2-3

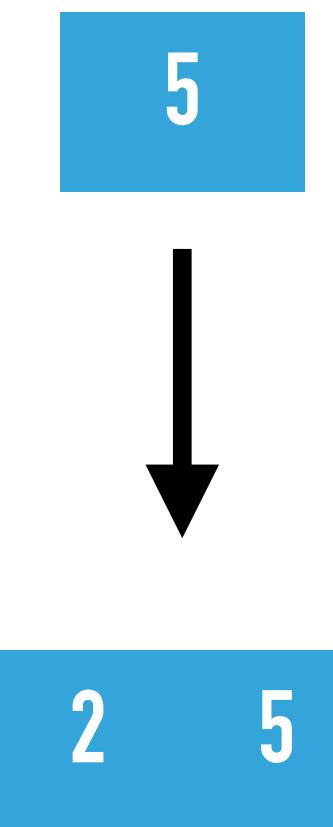
Arbre 2-3 contient soit :

- 2-noeud, avec 1 clé et deux liens
 - À gauche : clés plus petites
 - À droite : clés plus grandes
- 3-noeud, avec 2 clés et 3 liens
 - À gauche : clés plus petites
 - À droite : clés plus grandes
 - Au milieu : clés entre

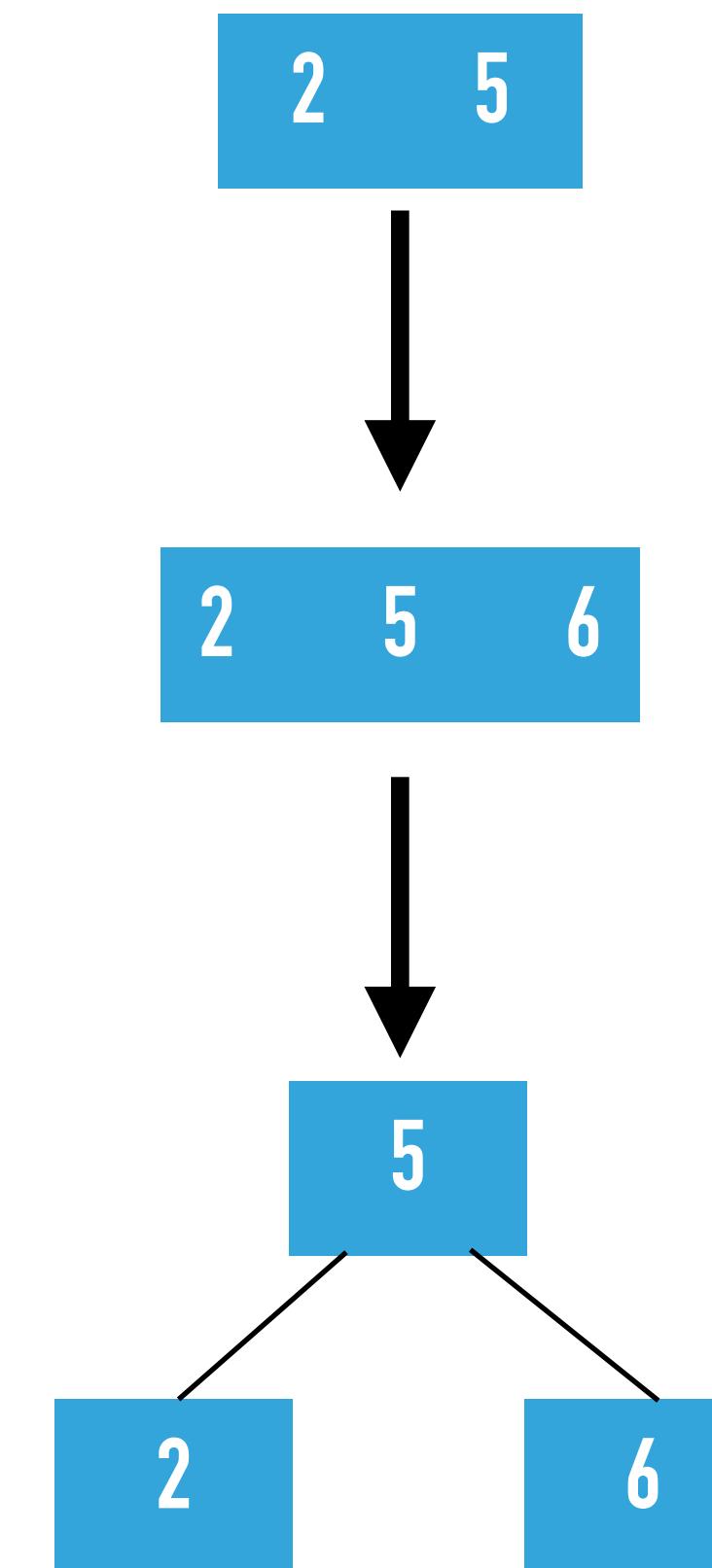


Insertions dans un arbre 2-3

Insertion dans un 2 noeud



Insertion dans un 3 noeud



Question 3.1.9 insertions

12

12, 5, 10, 3, 13, 14, 15, 17, 18, 15

Question 3.1.9 insertions

5 12

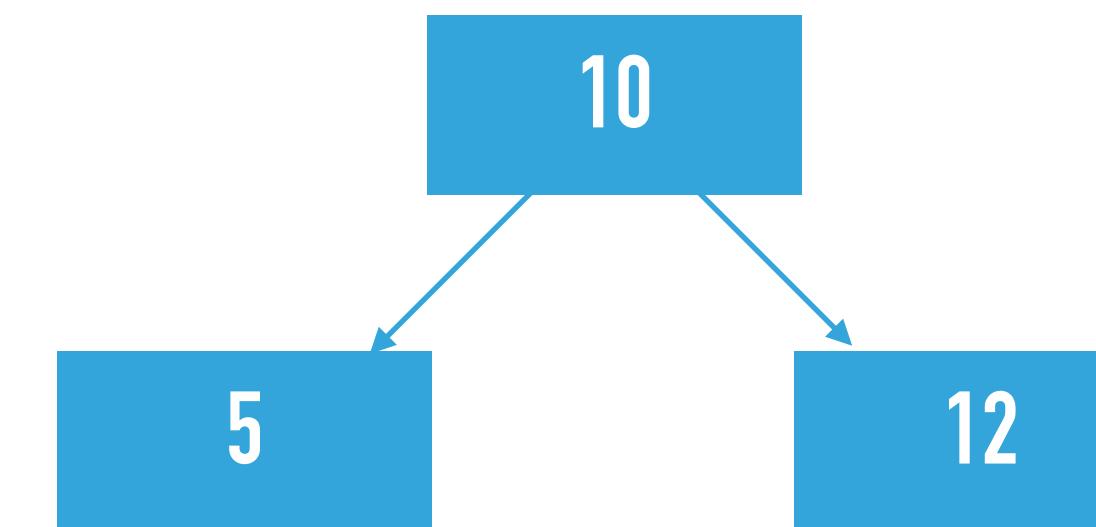
12, 5, 10, 3, 13, 14, 15, 17, 18, 15

Question 3.1.9 insertions

5 10 12

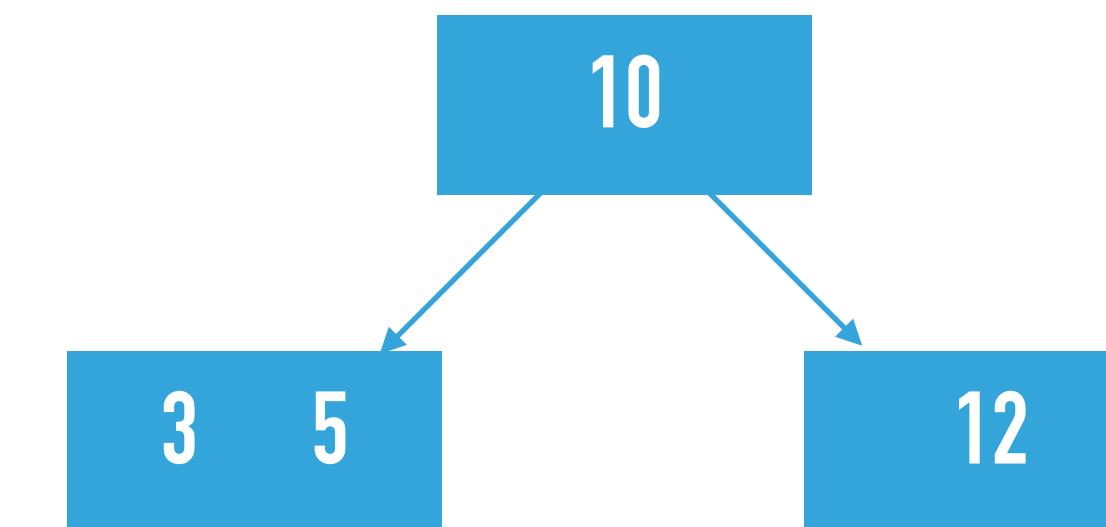
12, 5, 10, 3, 13, 14, 15, 17, 18, 15

Question 3.1.9 insertions



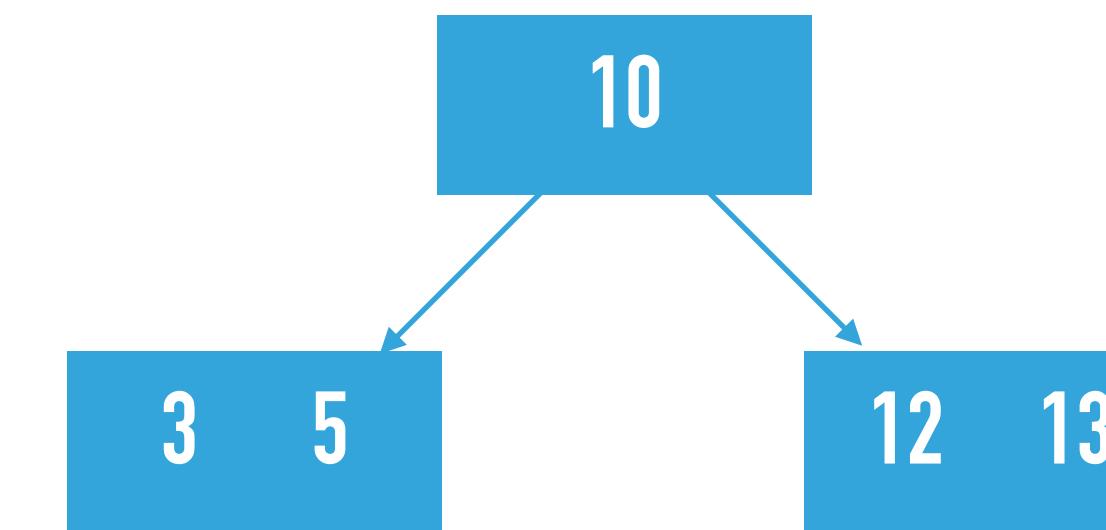
12, 5, 10, 3, 13, 14, 15, 17, 18, 15

Question 3.1.9 insertions



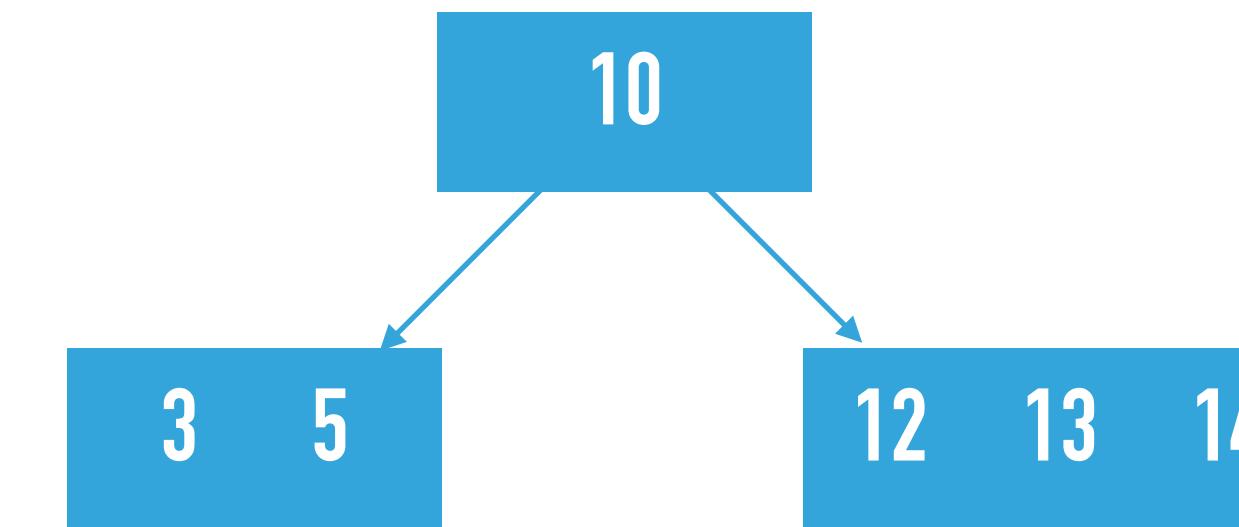
~~12, 5, 10, 3, 13, 14, 15, 17, 18, 15~~

Question 3.1.9 insertions



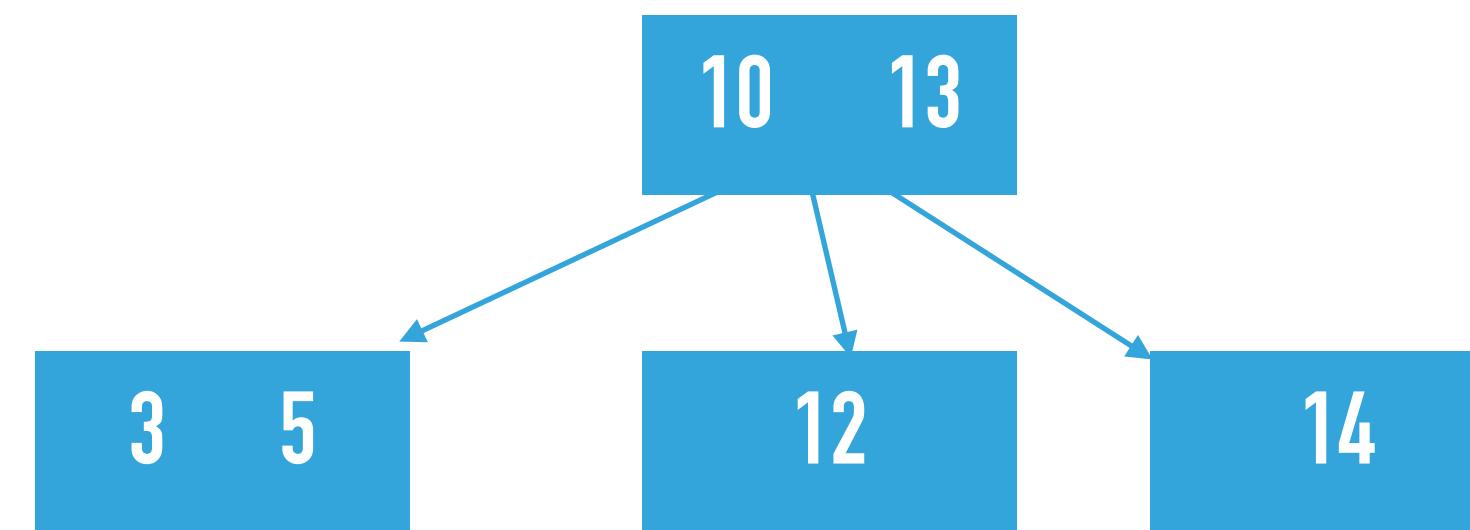
~~12, 5, 10, 3, 13, 14, 15, 17, 18, 15~~

Question 3.1.9 insertions



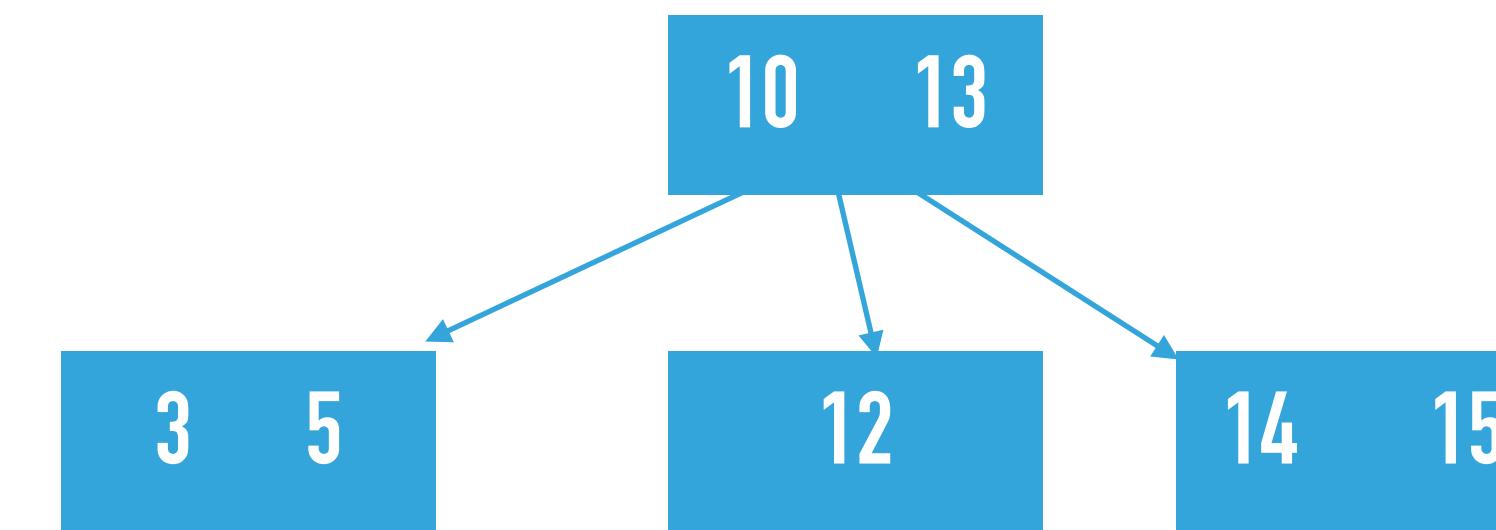
~~12, 5, 10, 3, 13, 14, 15, 17, 18, 15~~

Question 3.1.9 insertions



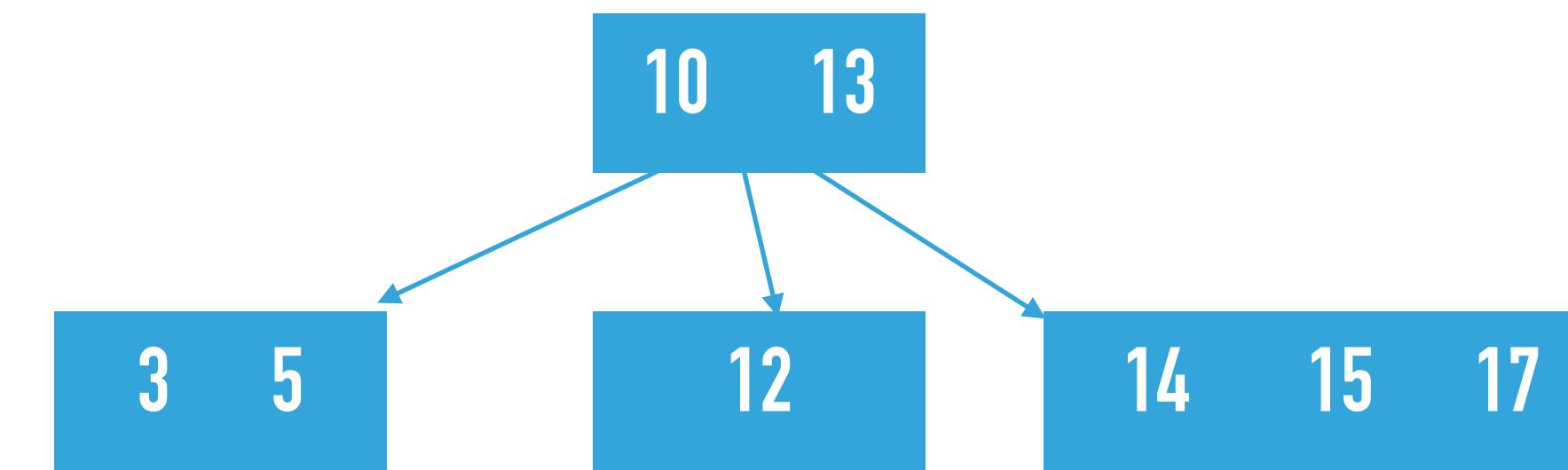
~~12, 5, 10, 3, 13, 14, 15, 17, 18, 15~~

Question 3.1.9 insertions



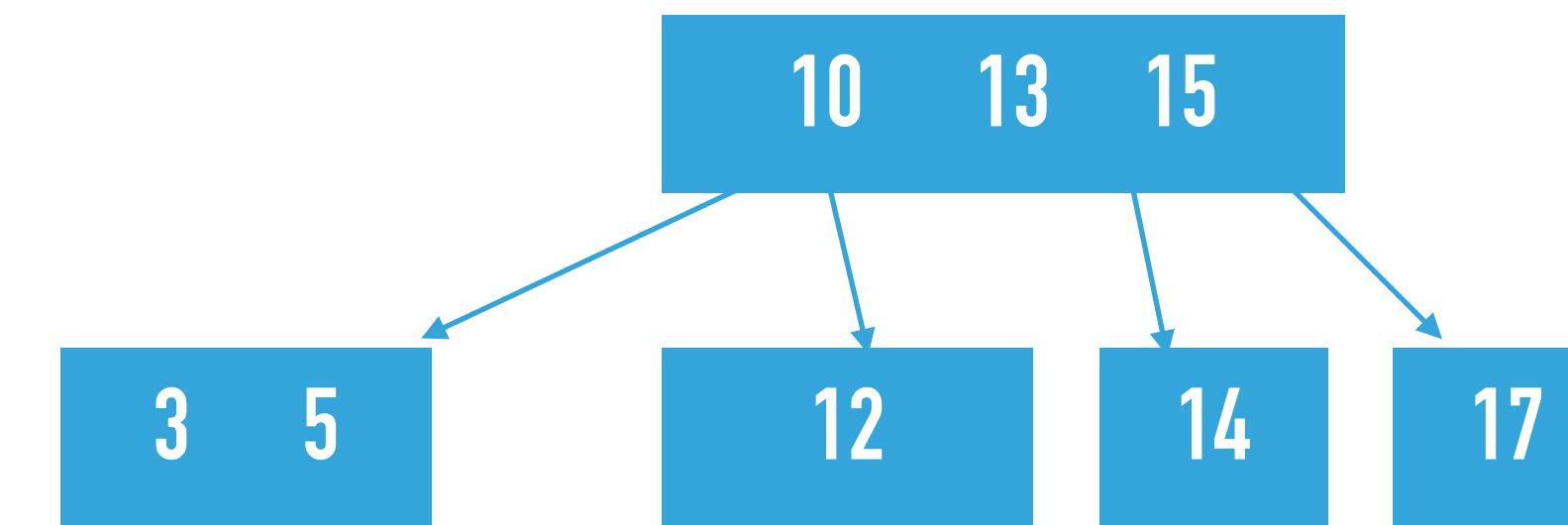
~~12, 5, 10, 3, 13, 14, 15, 17, 18, 15~~

Question 3.1.9 insertions



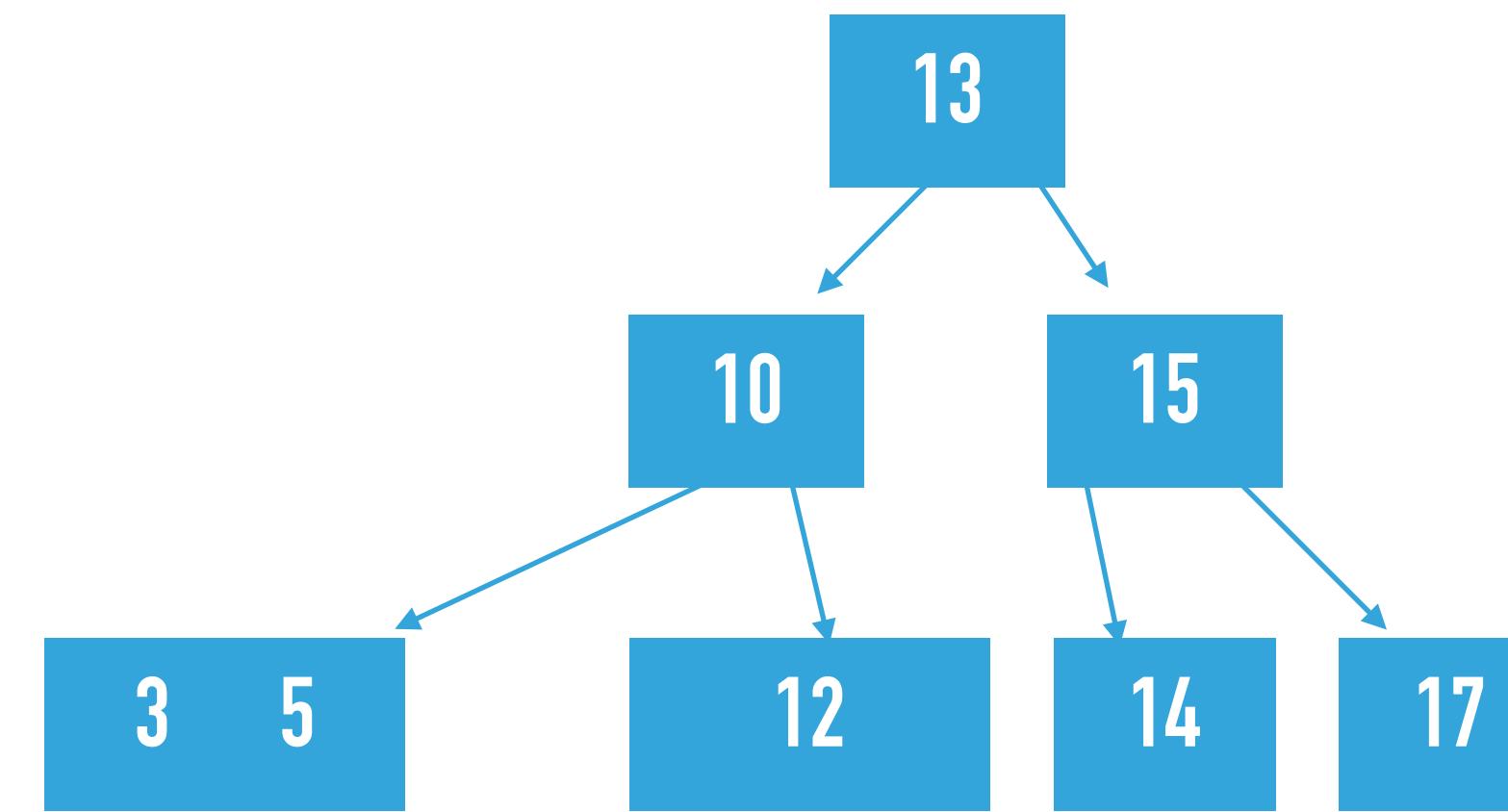
~~12, 5, 10, 3, 13, 14, 15, 17, 18, 15~~

Question 3.1.9 insertions



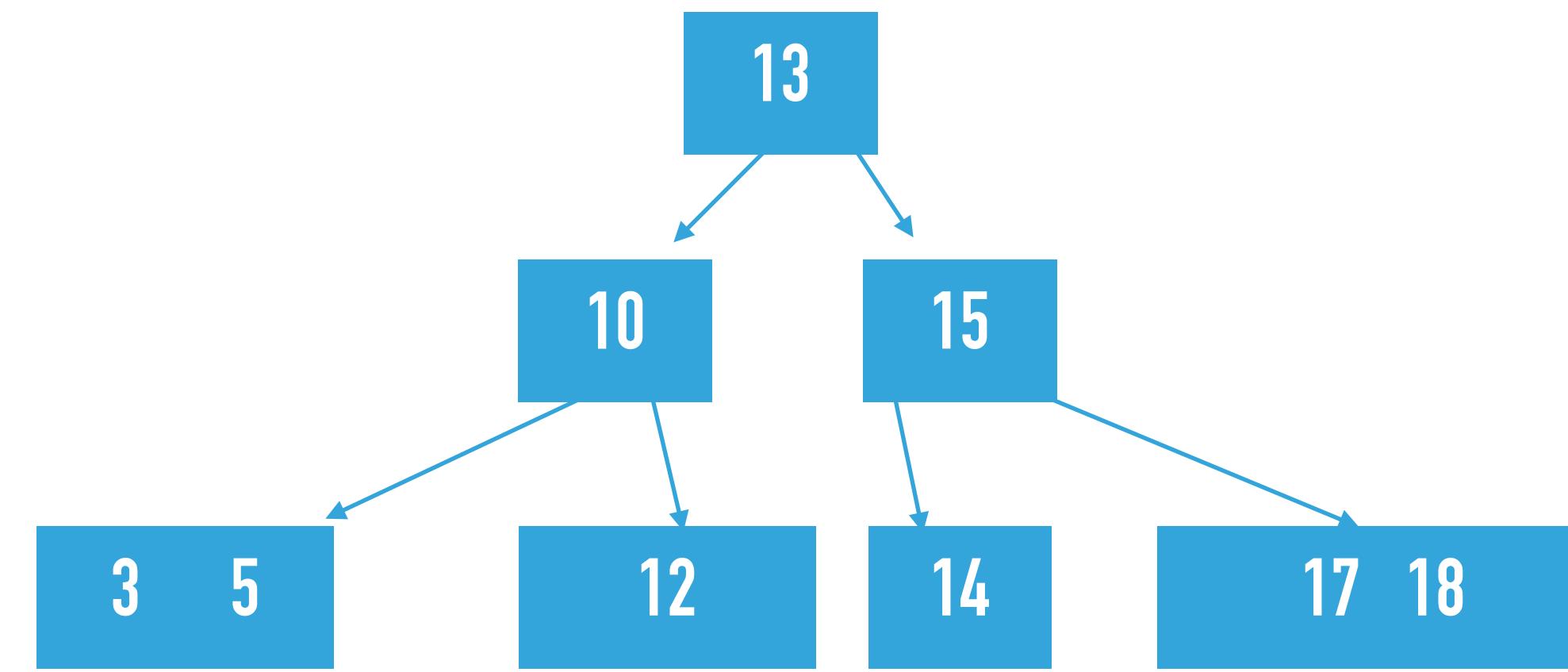
~~12, 5, 10, 3, 13, 14, 15, 17, 18, 15~~

Question 3.1.9 insertions



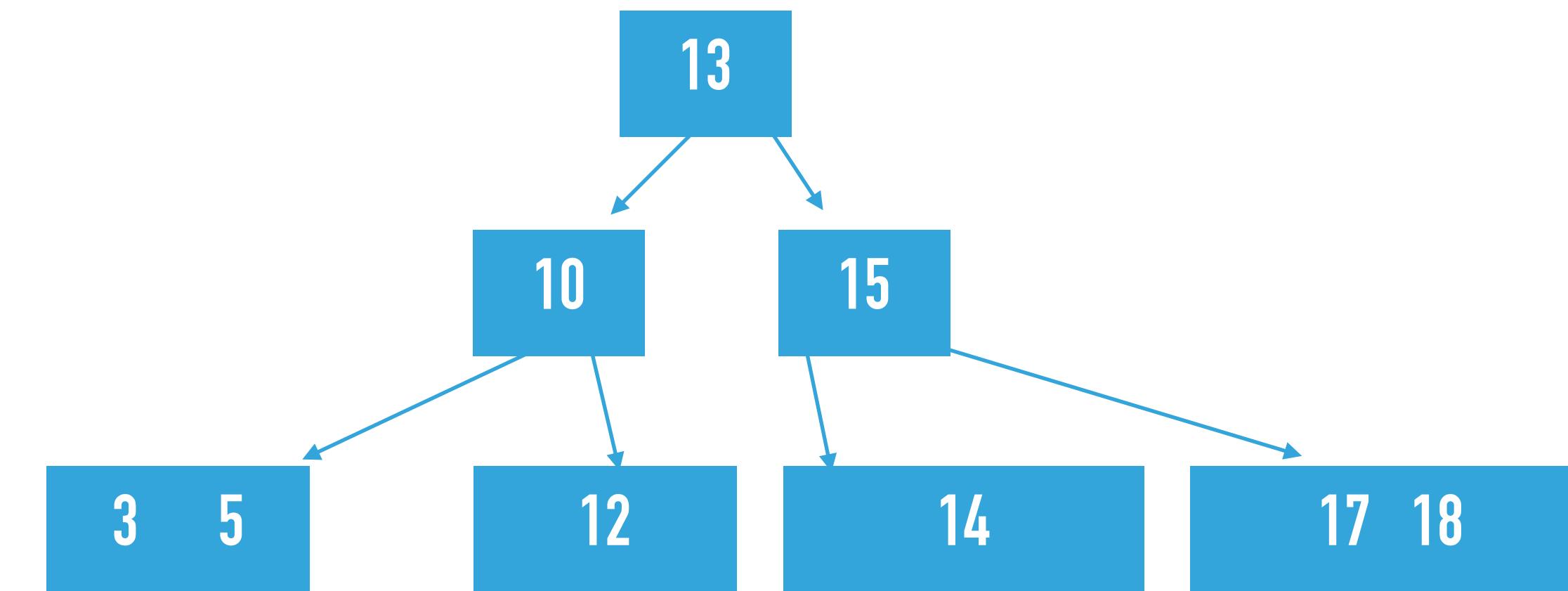
~~12, 5, 10, 3, 13, 14, 15, 17, 18, 15~~

Question 3.1.9 insertions



12, 5, 10, 3, 13, 14, 15, 17, 18, 15

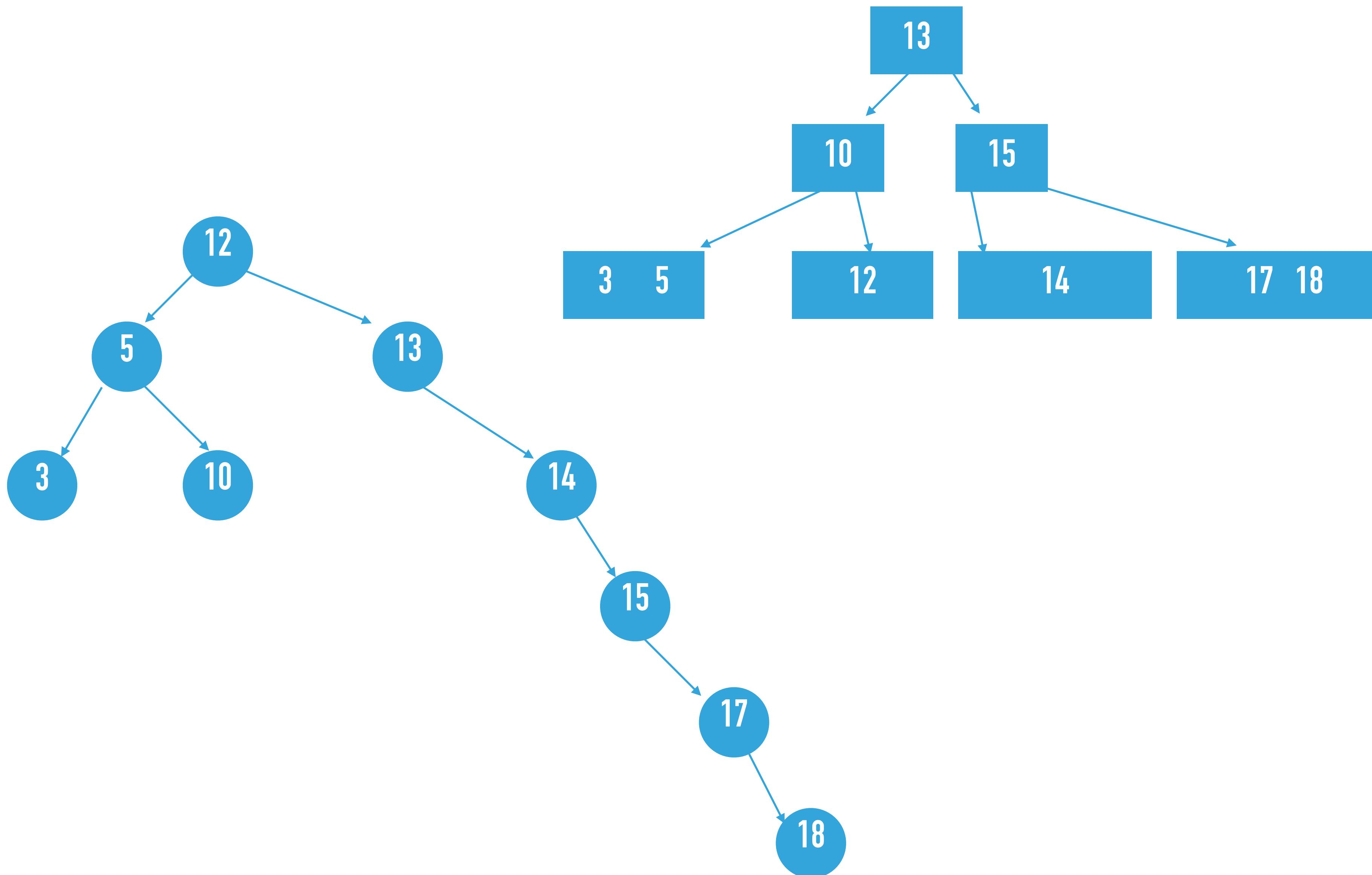
Question 3.1.9 insertions



15 est déjà dans l'arbre

~~12, 5, 10, 3, 13, 14, 15, 17, 18, 15~~

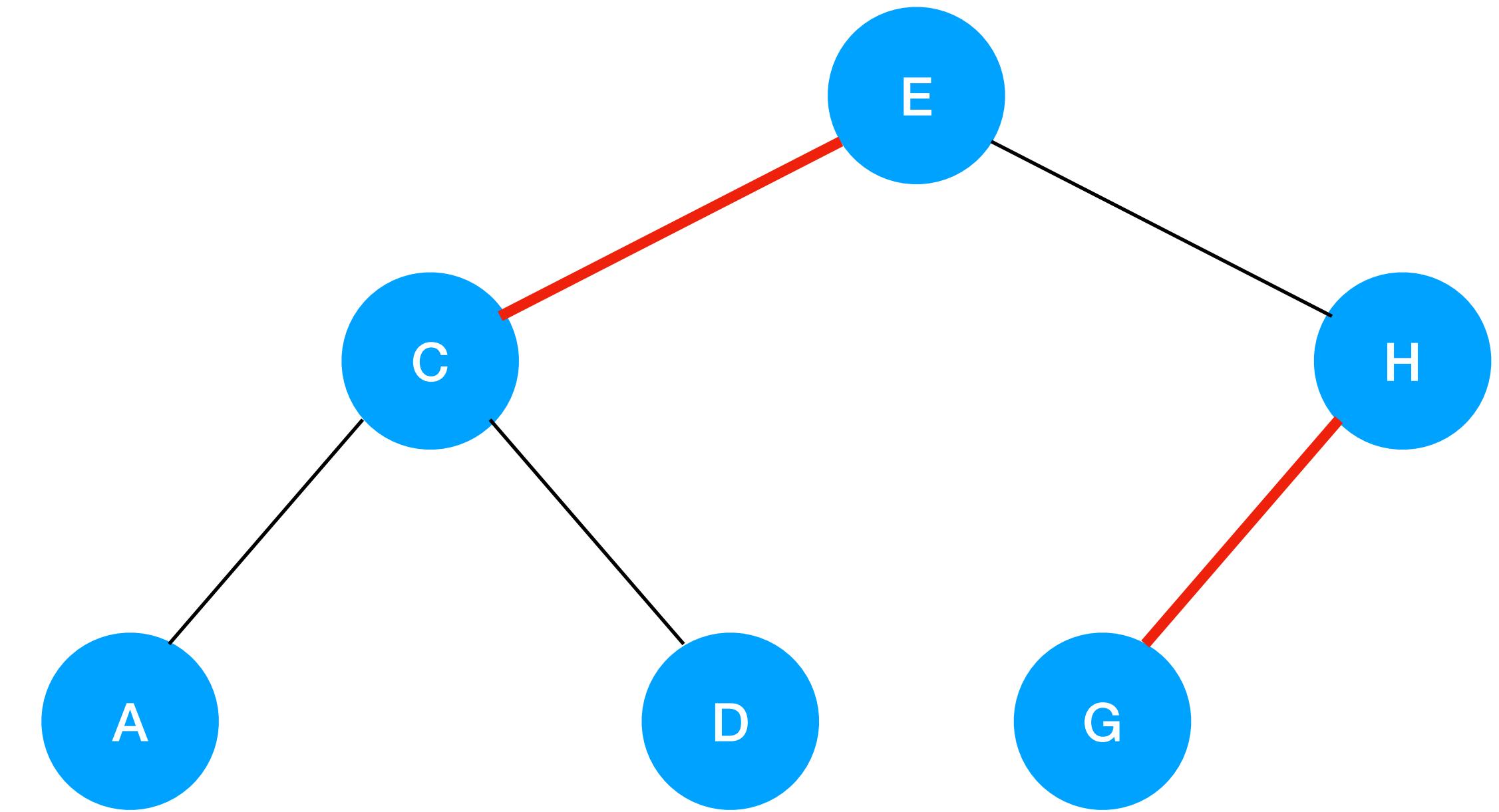
Question 3.1.9 insertions



Arbre Red-Black

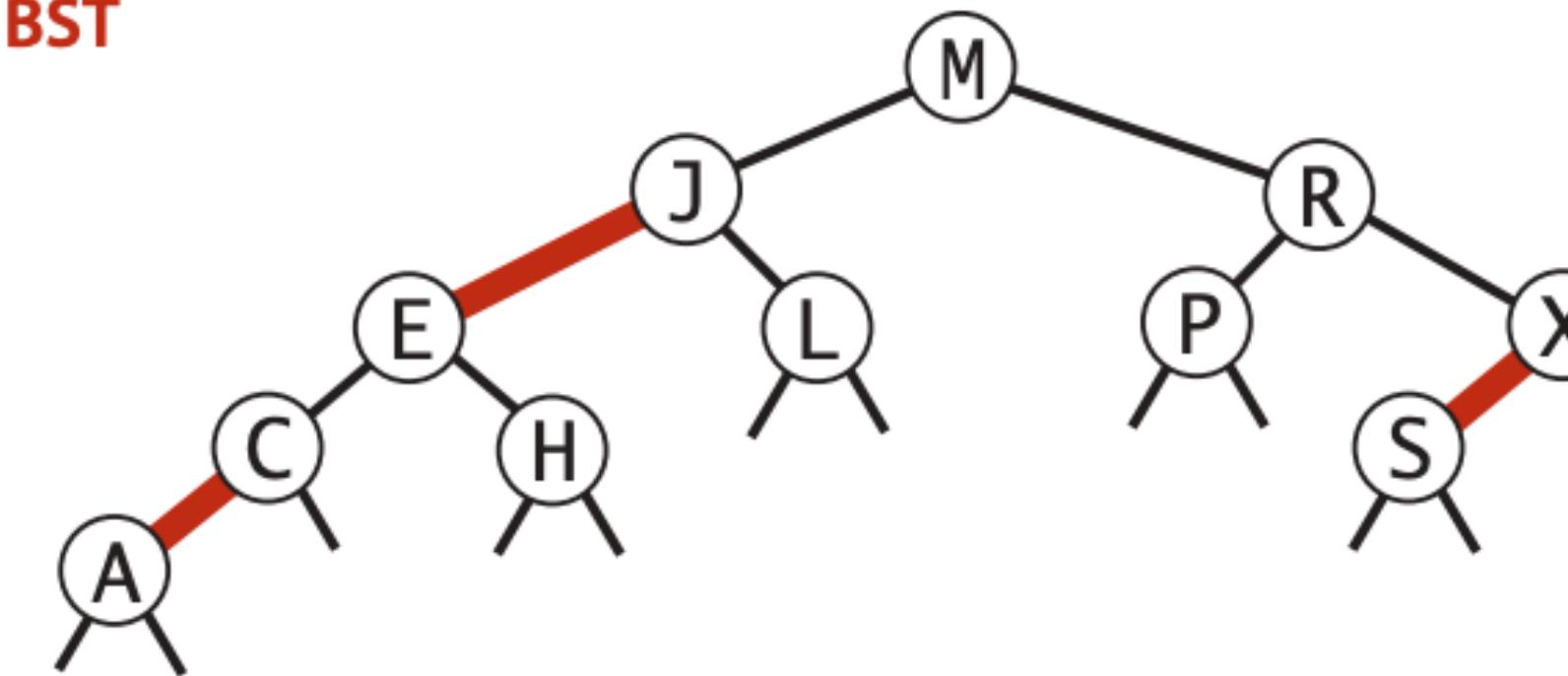
Arbre Red-Black :

- 2 types de liens (**rouge ou noir**)
- Liens **rouges** tjs à **gauche**
- **Pas** deux liens **rouges d'affilée**
- **Equilibre parfait** sur les liens **noirs**

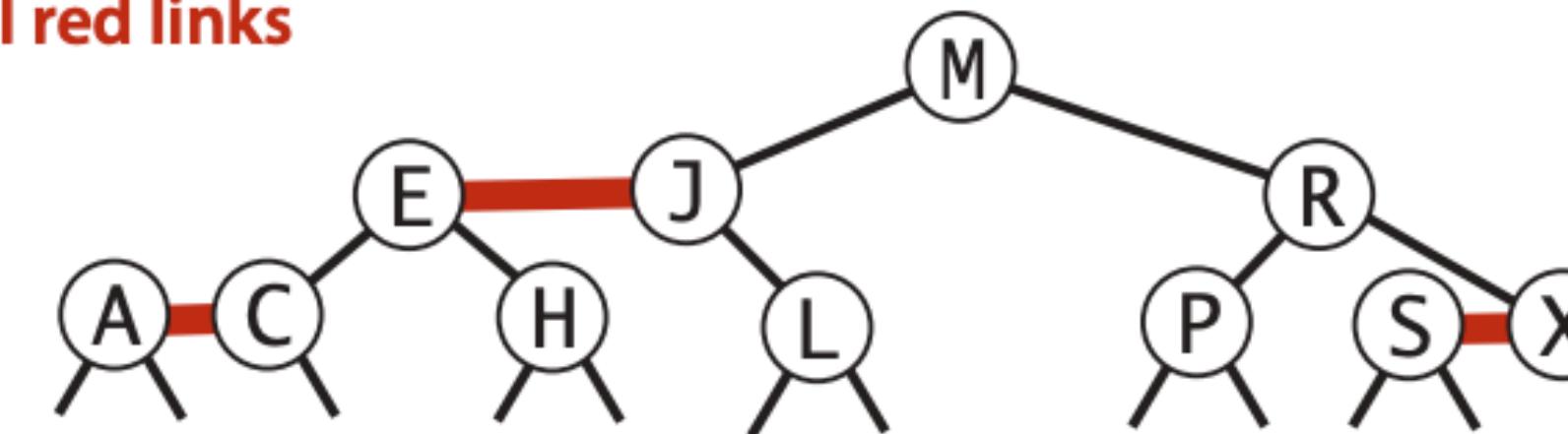


Correspondance 2-3 et Red-Black

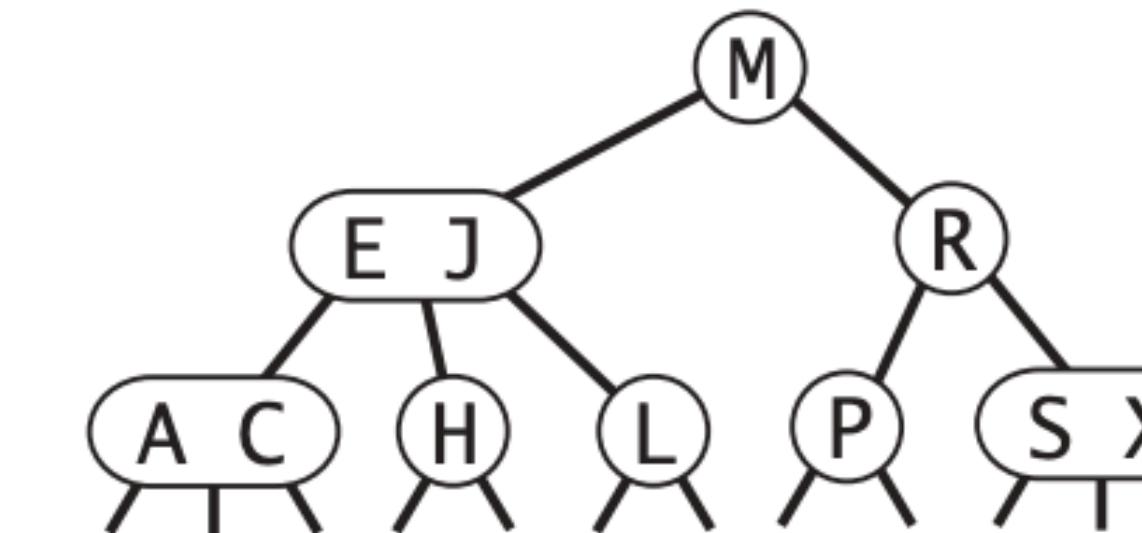
red-black BST



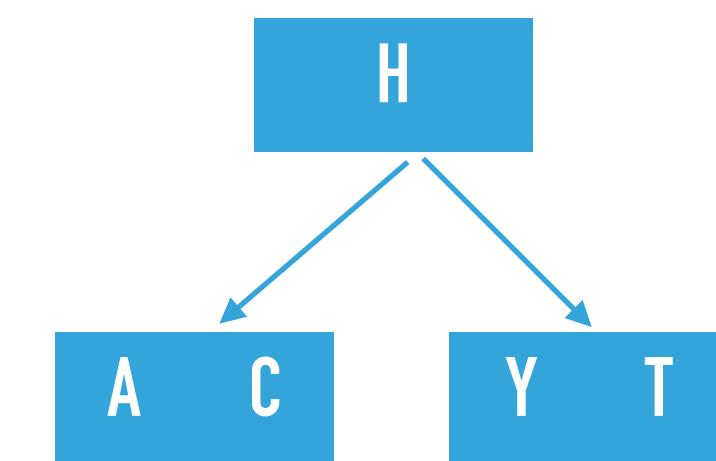
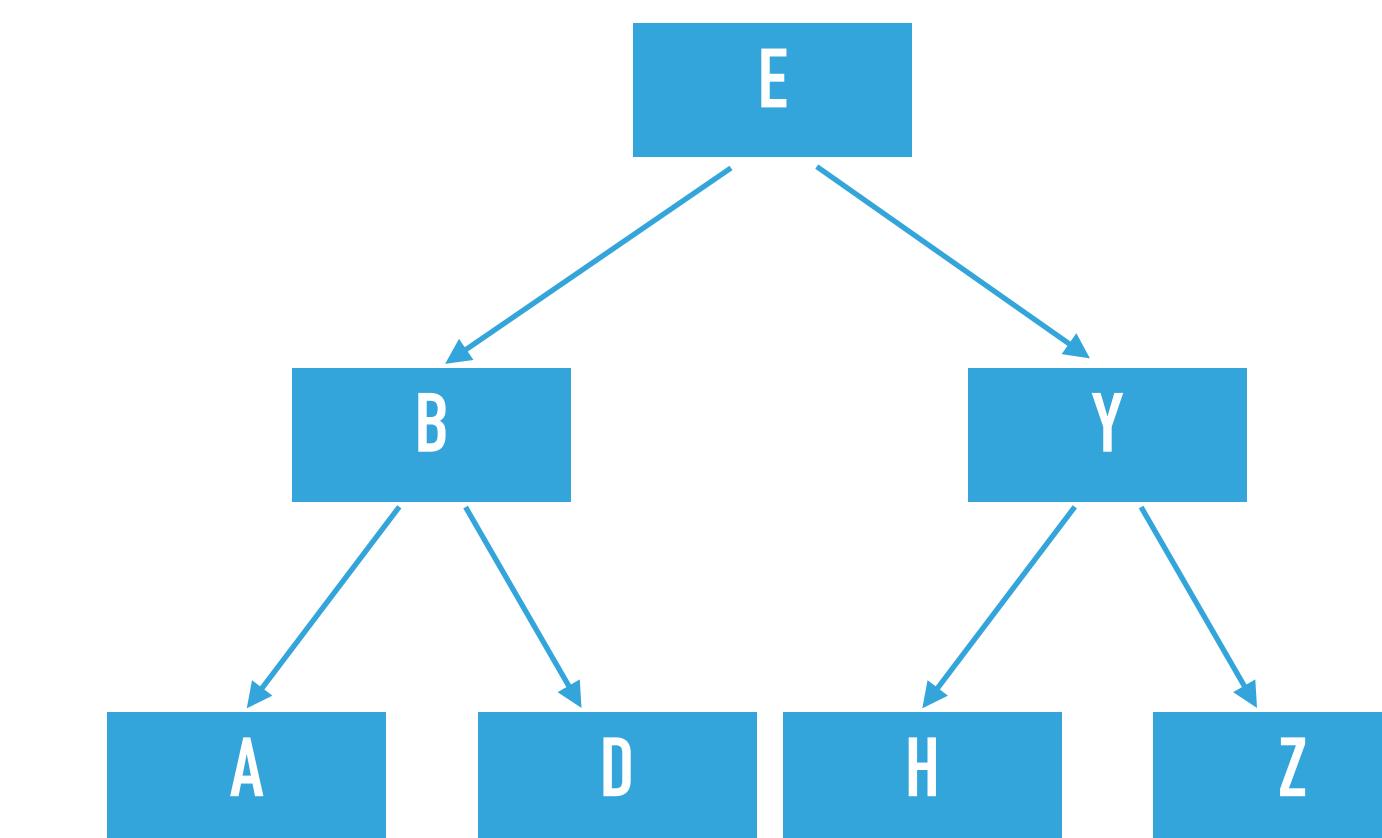
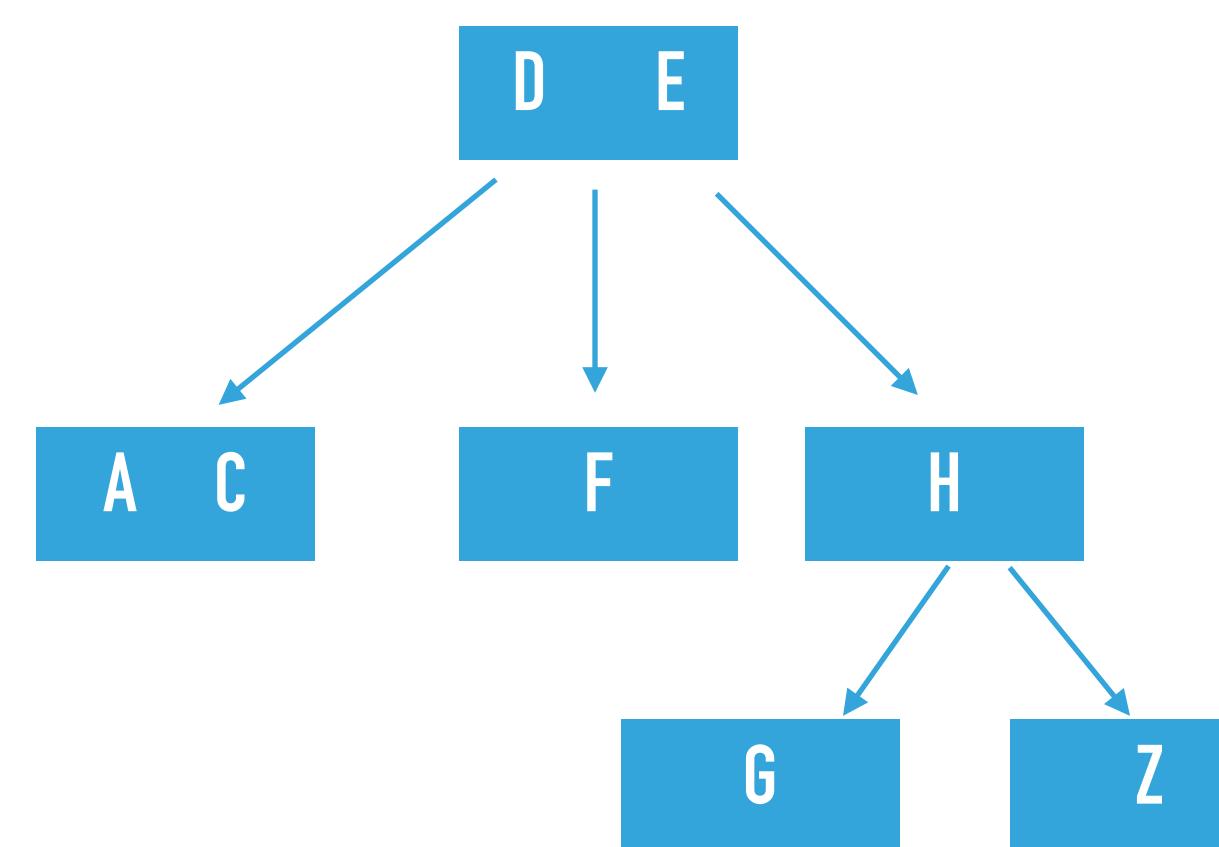
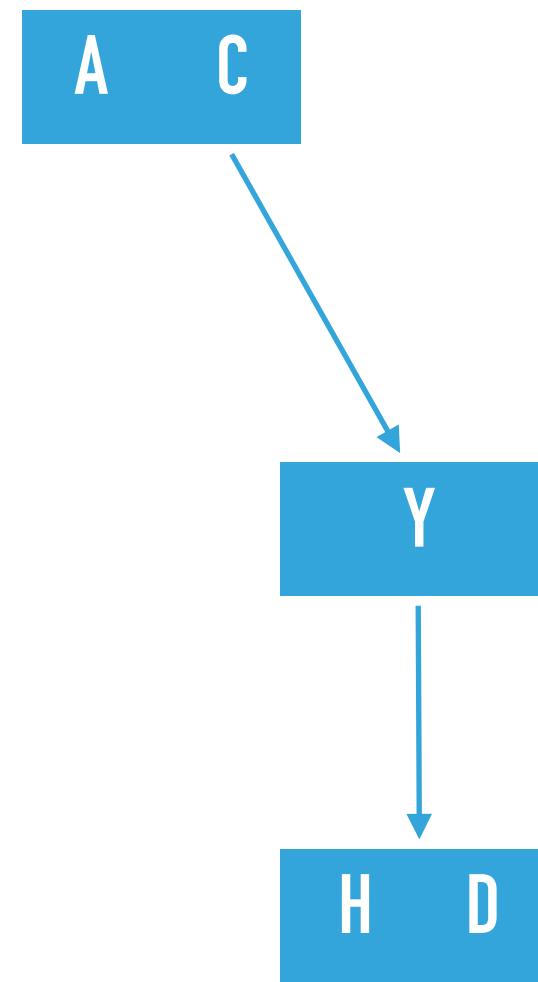
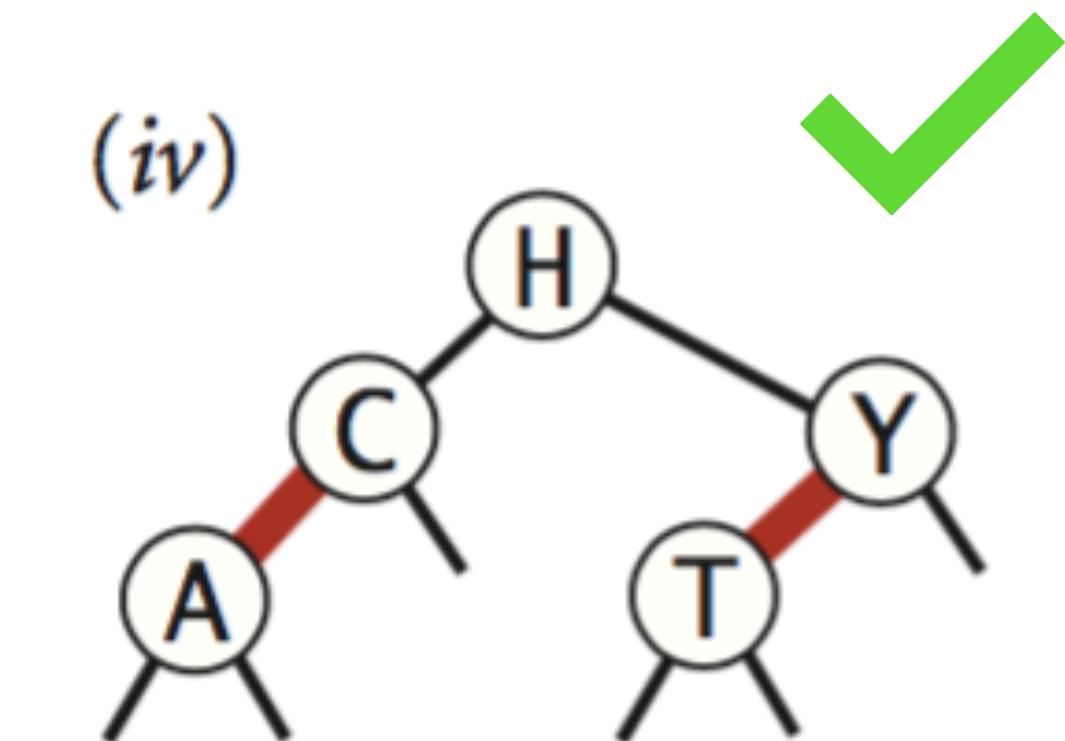
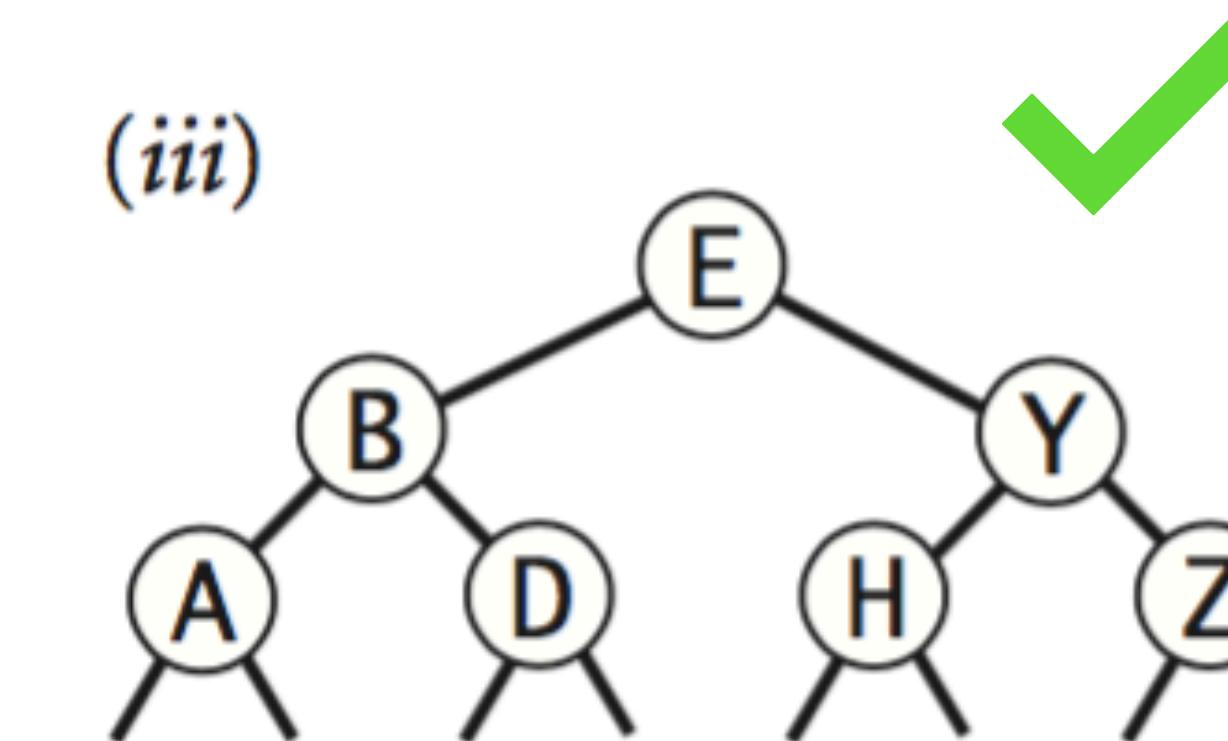
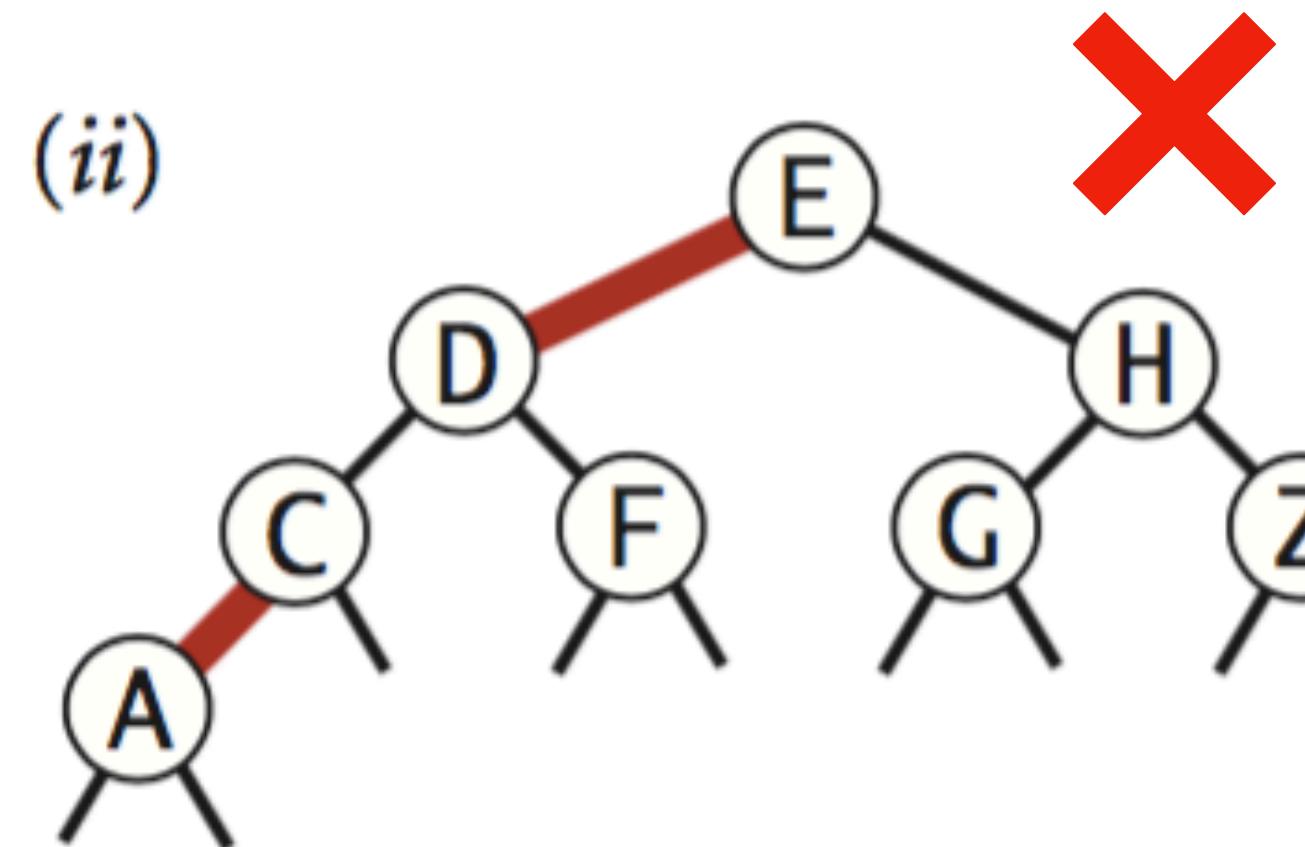
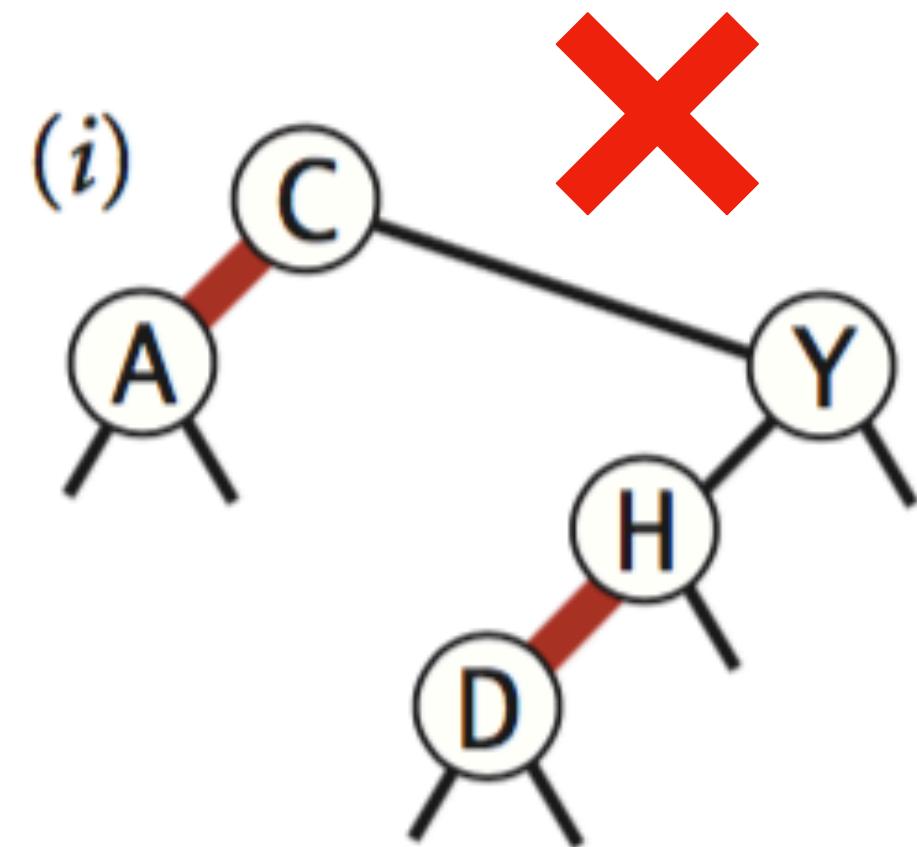
horizontal red links



2-3 tree



Question 3.1.9 RB?



Question 3.1.6 IS RED BLACK

```
private boolean is23() { return is23(root); }
```

```
private boolean is23(Node x) {  
    if (x == null)  
        return true;
```

//Est-ce que le lien à droite est rouge ?

```
if (isRed(x.right))  
    return false;
```

//Est-ce qu'il y a 2 liens rouges d'affilée ?

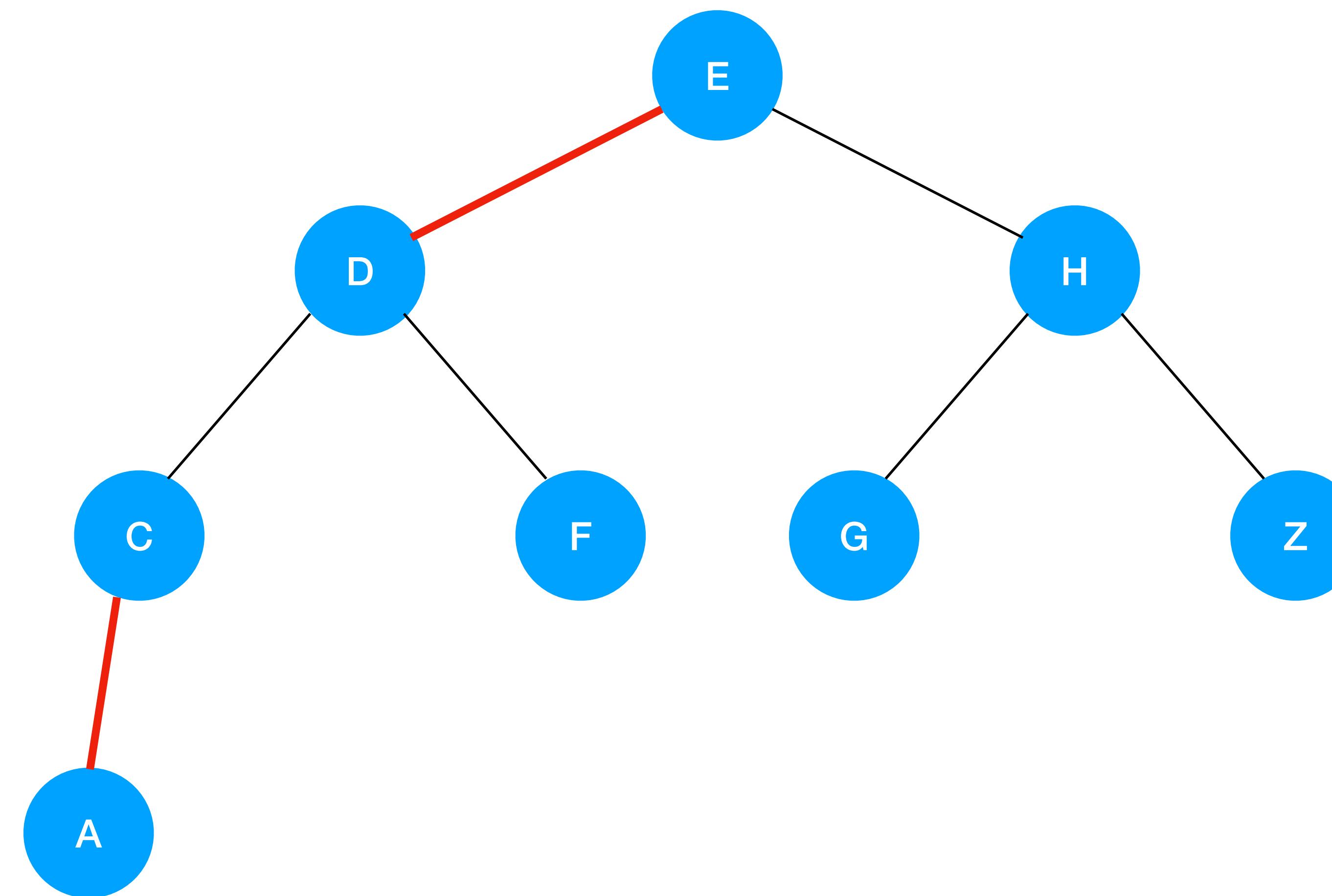
```
if (x != root && isRed(x) && isRed(x.left))  
    return false;  
return is23(x.left) && is23(x.right);  
}
```

Question 3.1.6 IS RED BLACK

```
private boolean isBalanced() {  
    // nombre de liens noirs de la racine à la clé min  
    int black = 0;  
    Node x = root;  
    while (x != null) {  
        if (!isRed(x)) black++;  
        x = x.left;  
    }  
    // Est-ce qu'on retrouve ce nombre de liens pour chaque feuille ?  
    return isBalanced(root, black);  
}
```

```
private boolean isBalanced(Node x, int black) {  
    if (x == null) return black == 0;  
    if (!isRed(x)) black--;  
    return isBalanced(x.left, black) && isBalanced(x.right, black);  
}
```

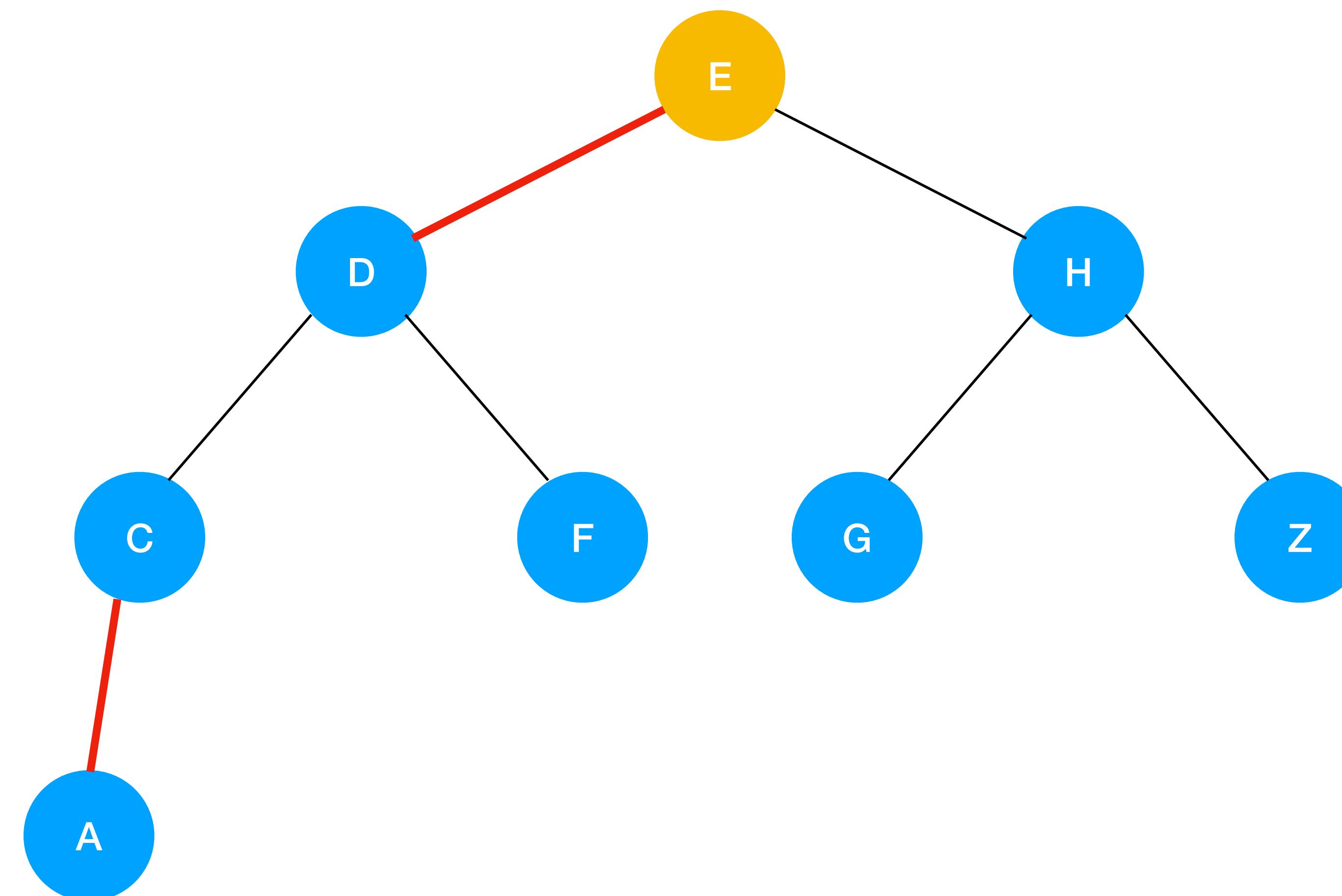
Question 3.1.6 IS RED BLACK



Node =
Black = 0

`int black = 0;`
`Node x = root;`

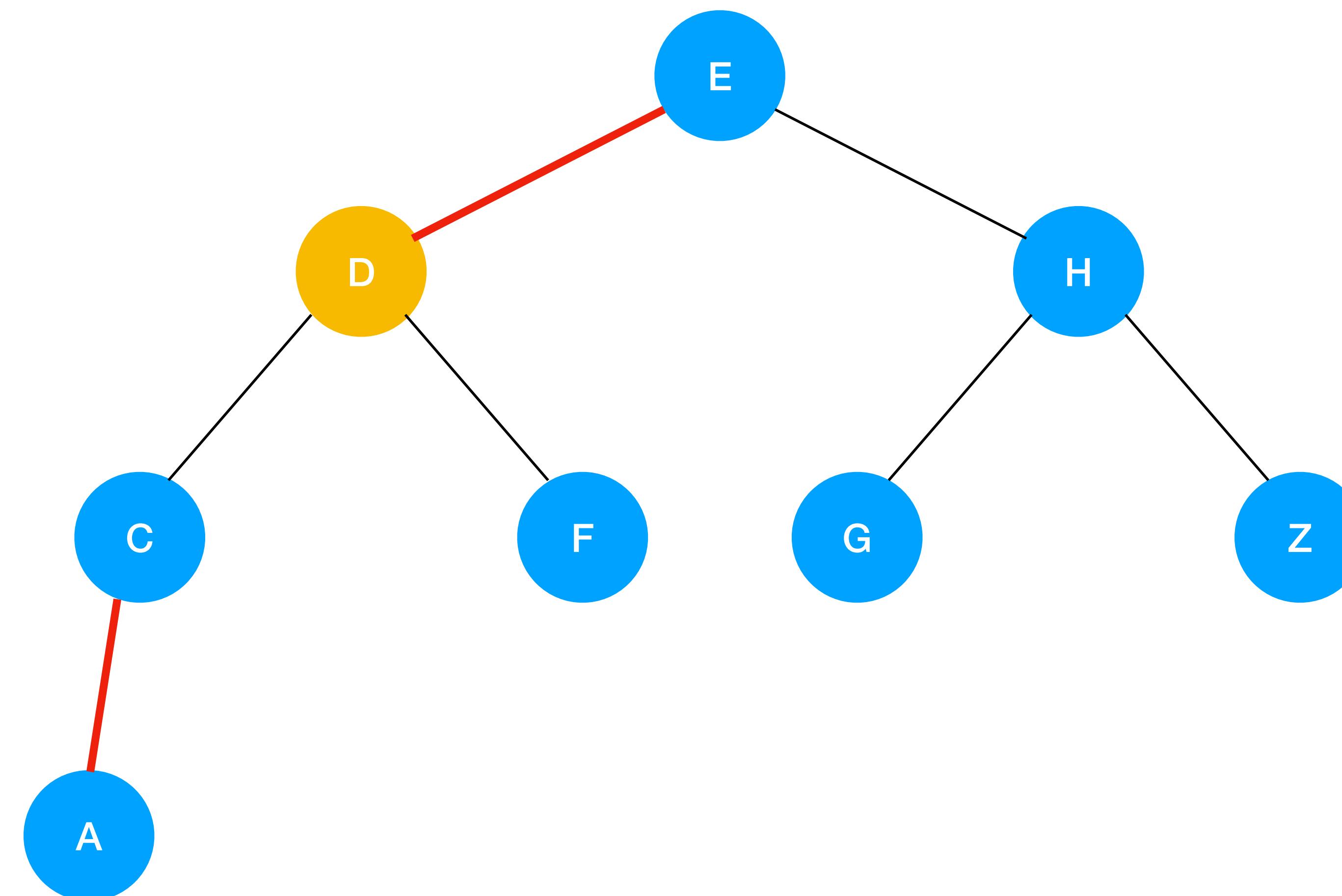
Question 3.1.6 IS RED BLACK



Node = E
Black = 1

```
while (x != null) {  
    if (!isRed(x)) black++;  
    x = x.left;  
}
```

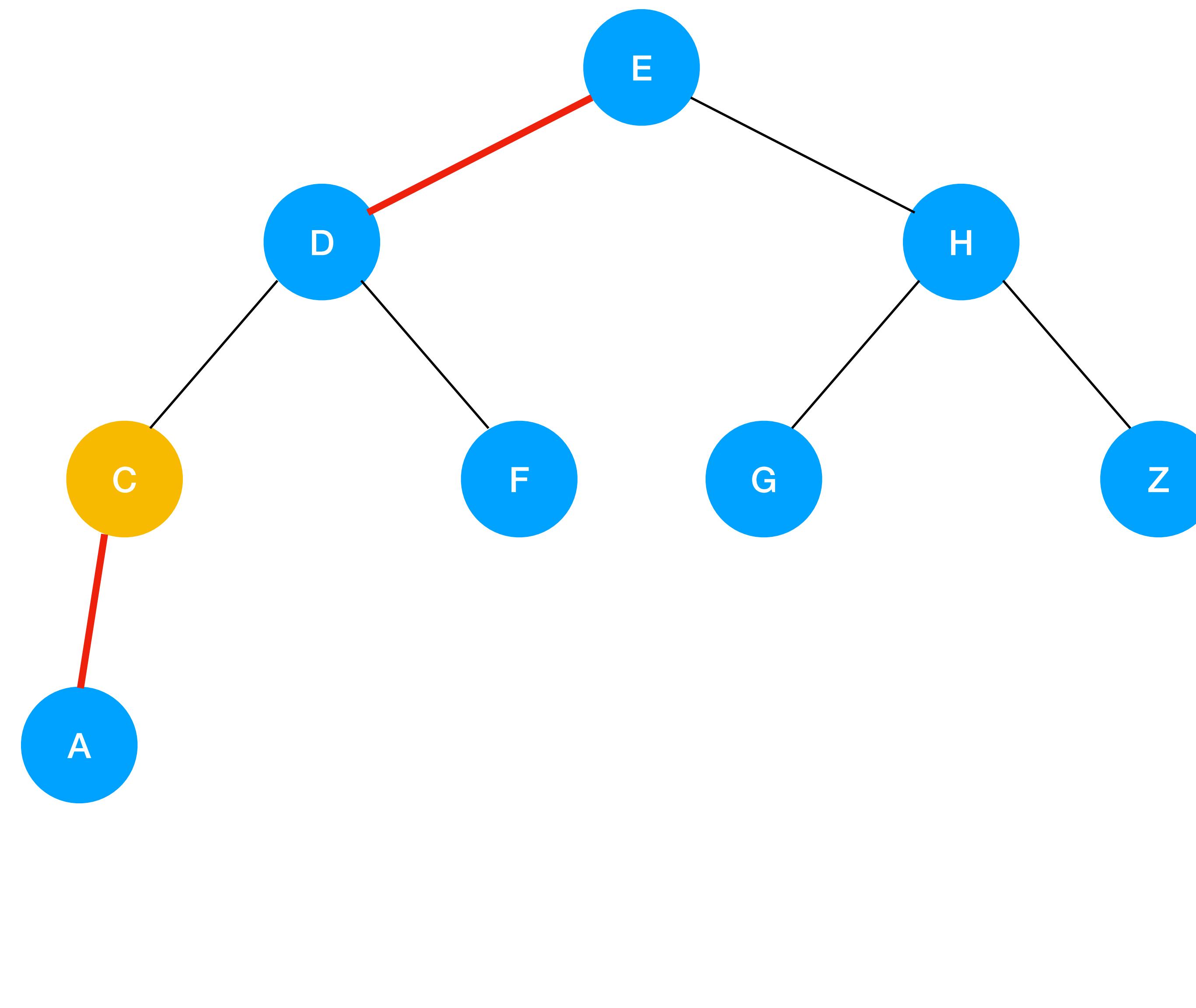
Question 3.1.6 IS RED BLACK



Node = D
Black = 1

```
while (x != null) {  
    if (!isRed(x)) black++;  
    x = x.left;  
}
```

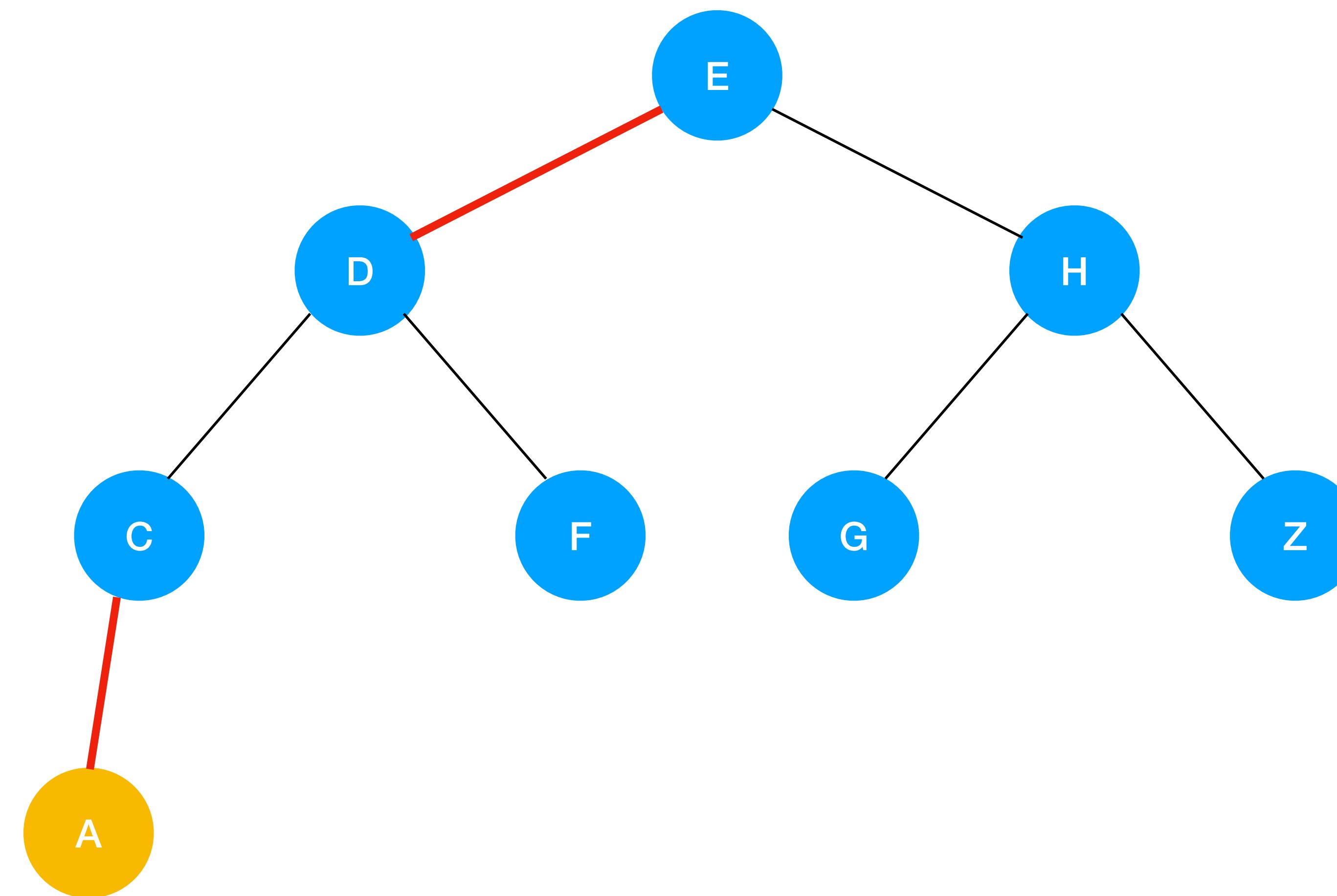
Question 3.1.6 IS RED BLACK



Node = C
Black = 2

```
while (x != null) {  
    if (!isRed(x)) black++;  
    x = x.left;  
}
```

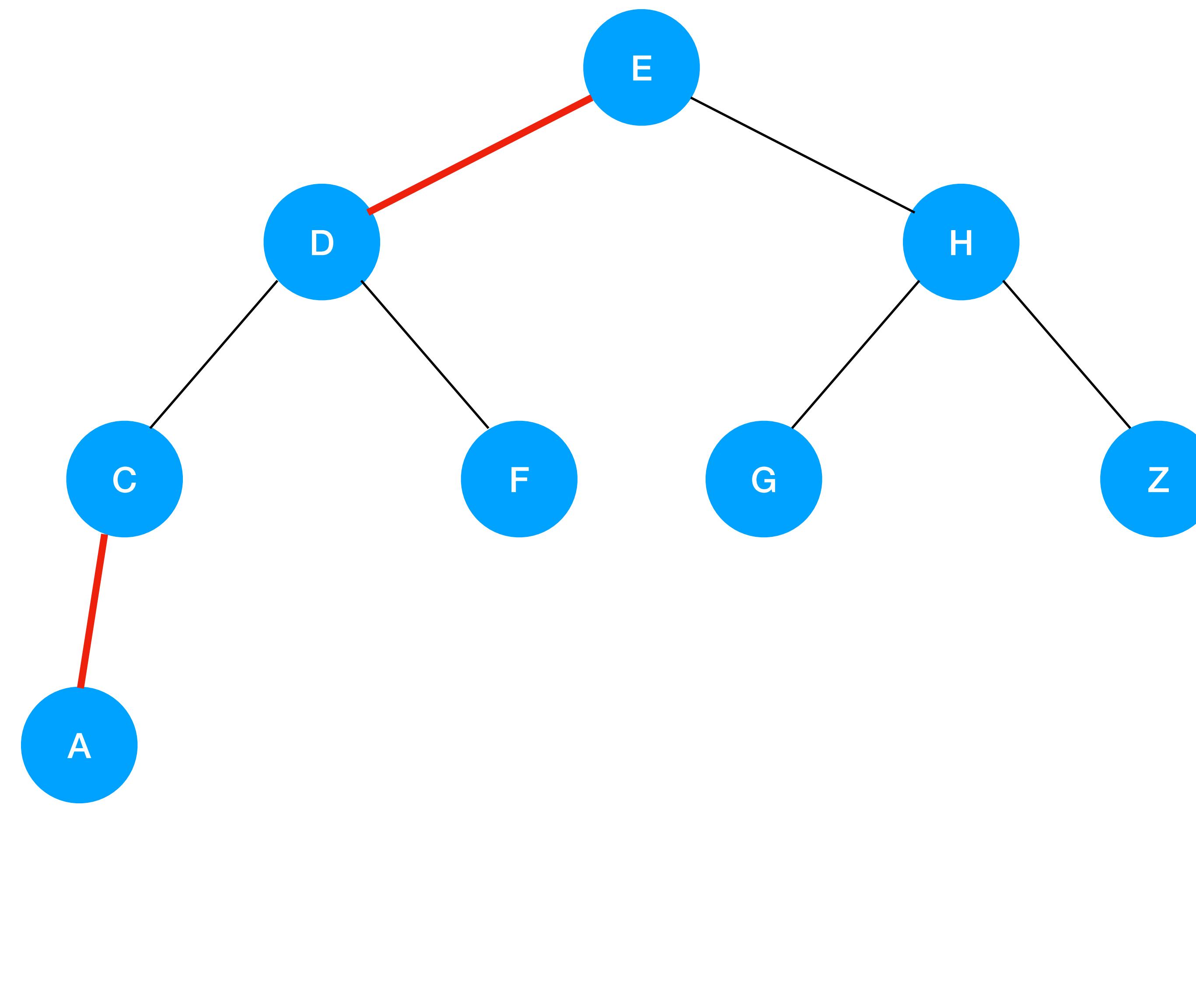
Question 3.1.6 IS RED BLACK



Node = A
Black = 2

```
while (x != null) {  
    if (!isRed(x)) black++;  
    x = x.left;  
}
```

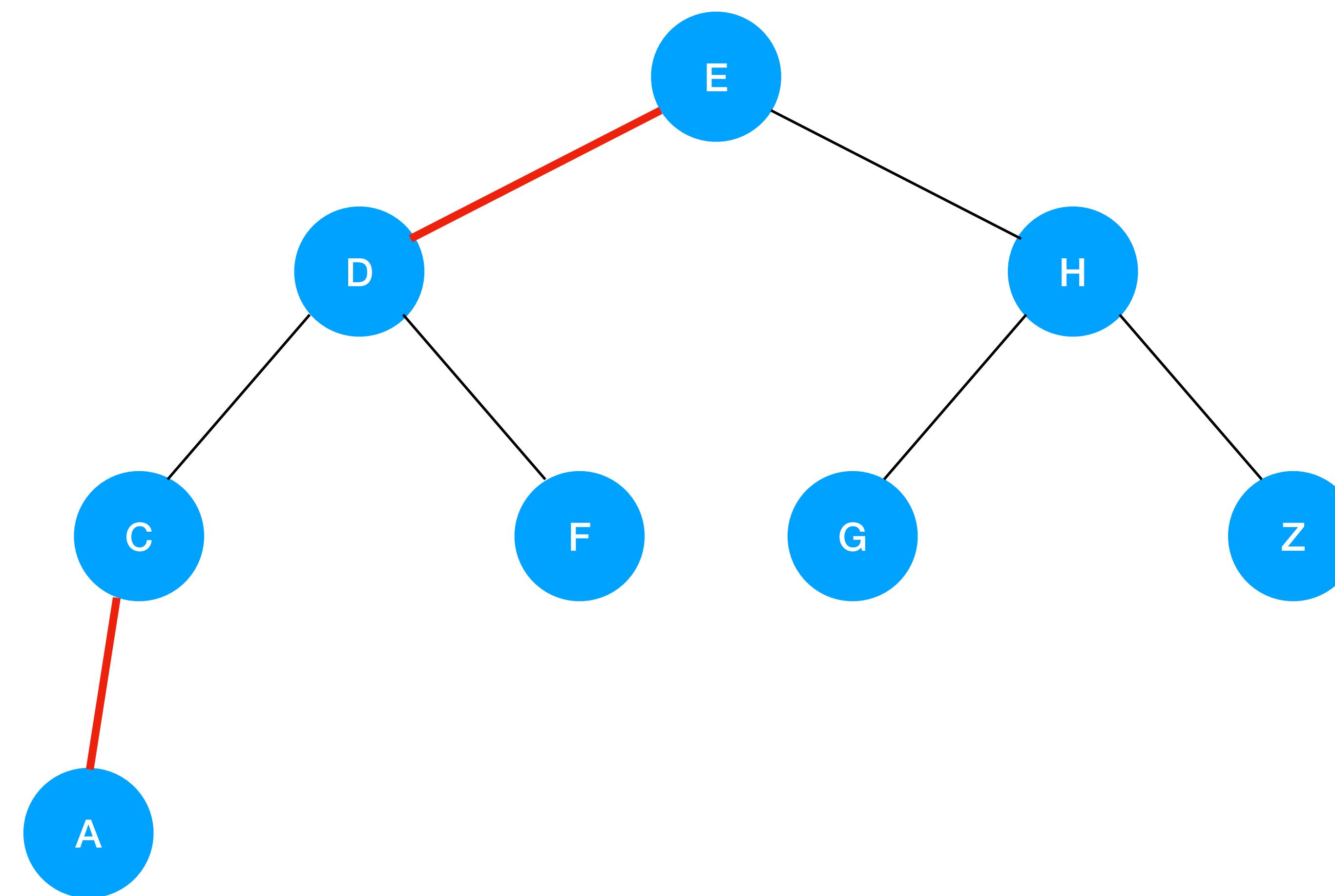
Question 3.1.6 IS RED BLACK



Node = null
Black = 2

```
while (x != null) {  
    if (!isRed(x)) black++;  
    x = x.left;  
}
```

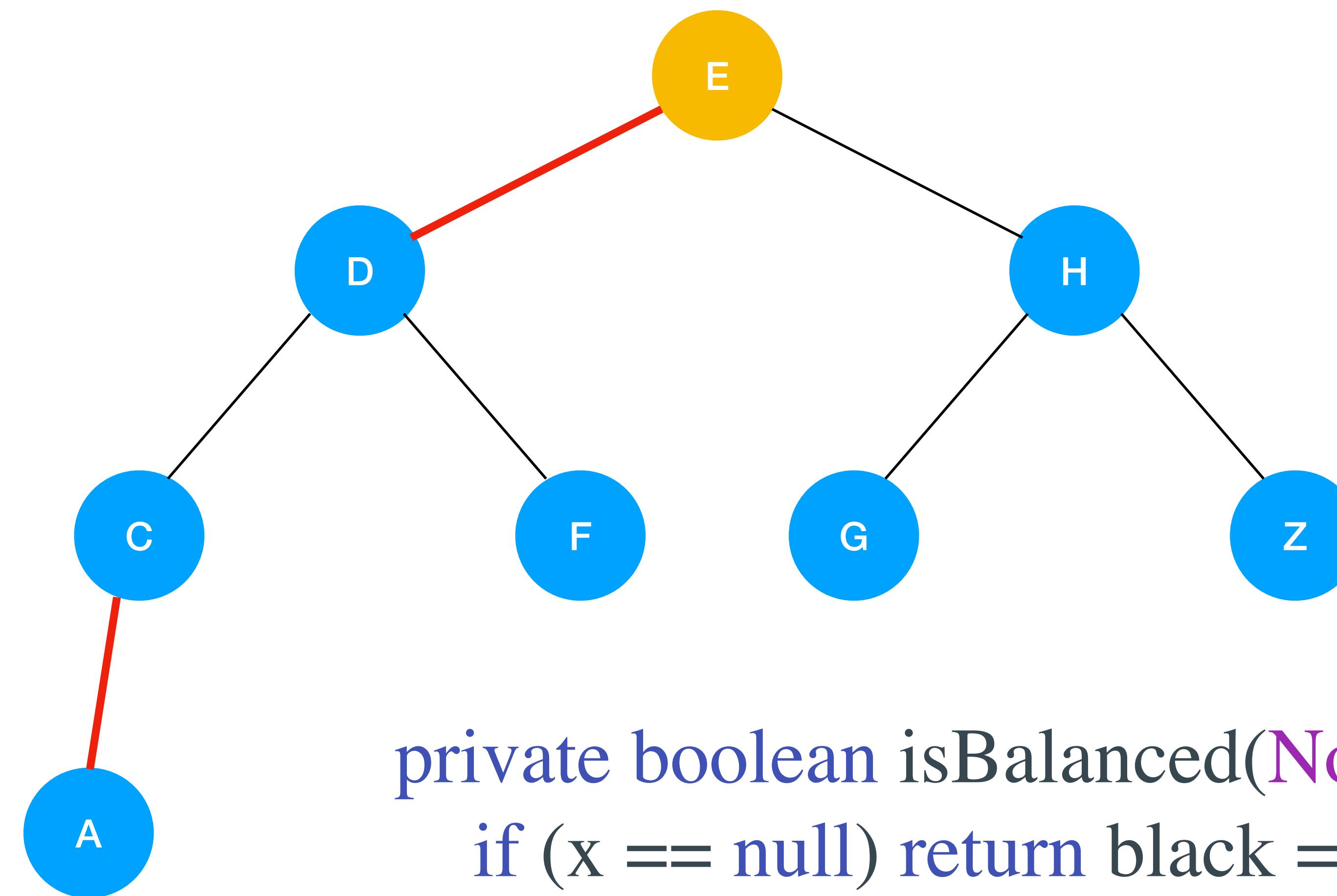
Question 3.1.6 IS RED BLACK



**Node =
Black = 2**

return isBalanced(root, black);

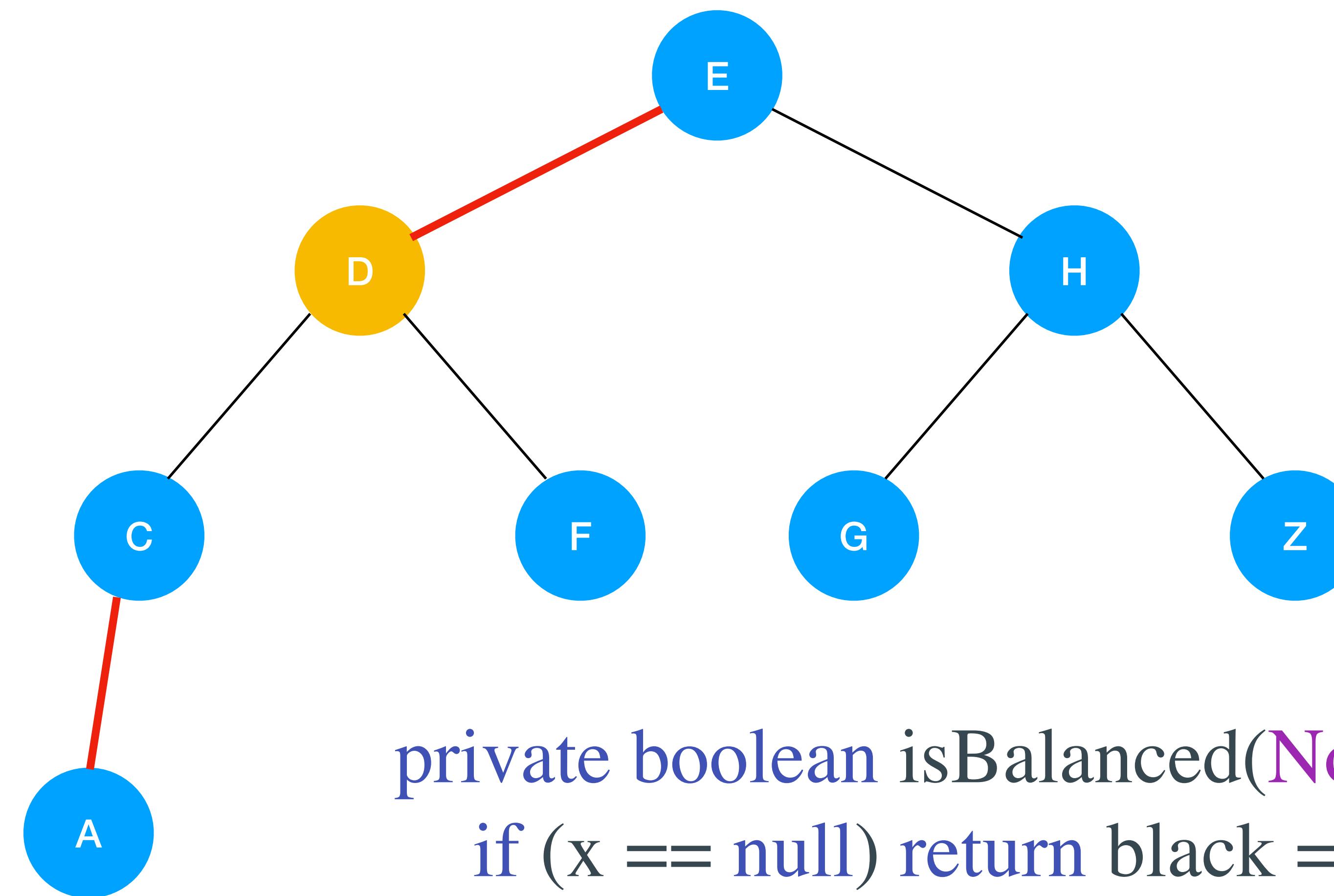
Question 3.1.6 IS RED BLACK



Node = E
Black = 1

```
private boolean isBalanced(Node x, int black) {  
    if (x == null) return black == 0;  
    if (!isRed(x)) black--;  
    return isBalanced(x.left, black) && isBalanced(x.right, black);  
}
```

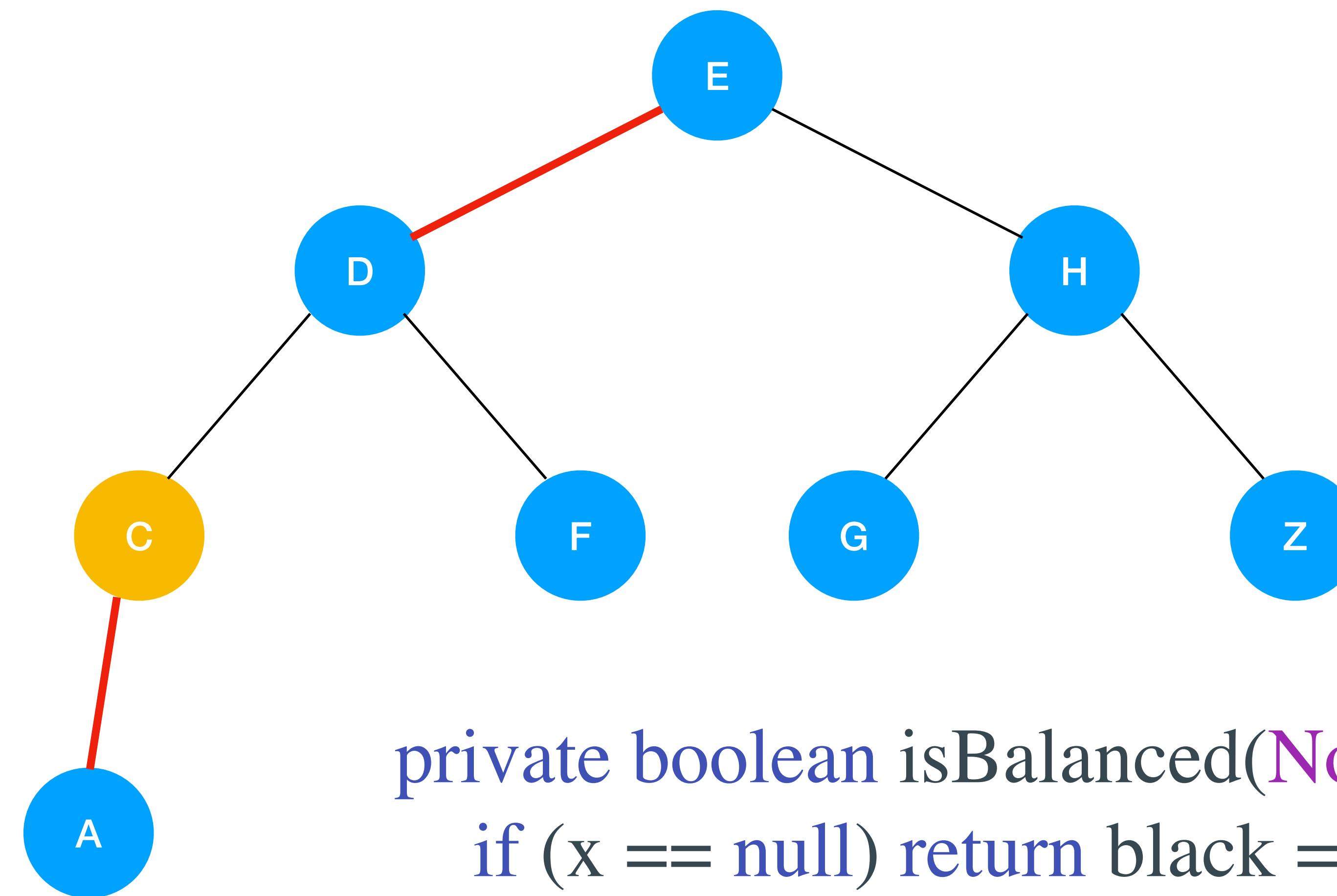
Question 3.1.6 IS RED BLACK



Node = D
Black = 1

```
private boolean isBalanced(Node x, int black) {  
    if (x == null) return black == 0;  
    if (!isRed(x)) black--;  
    return isBalanced(x.left, black) && isBalanced(x.right, black);  
}
```

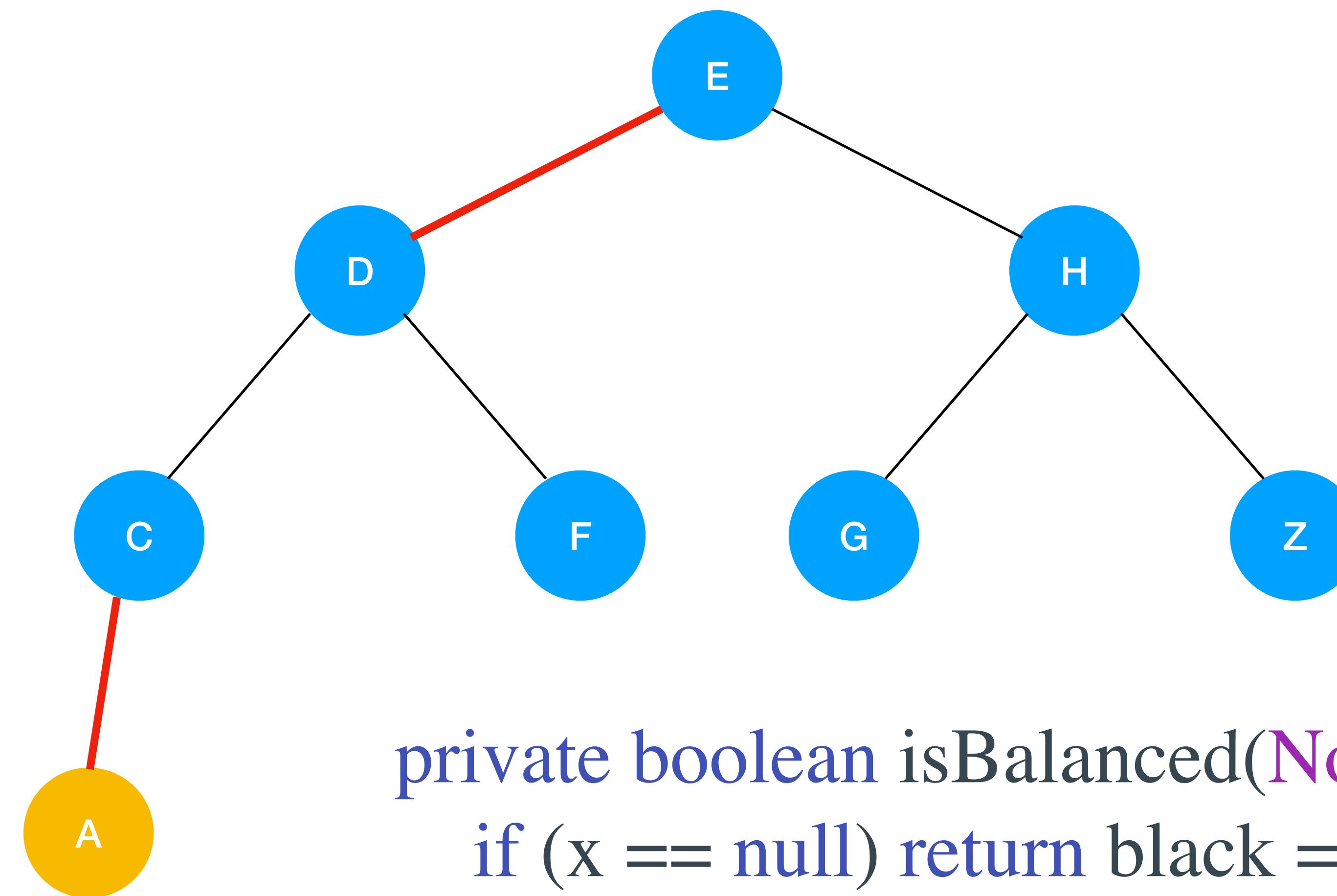
Question 3.1.6 IS RED BLACK



Node = C
Black = 0

```
private boolean isBalanced(Node x, int black) {  
    if (x == null) return black == 0;  
    if (!isRed(x)) black--;  
    return isBalanced(x.left, black) && isBalanced(x.right, black);  
}
```

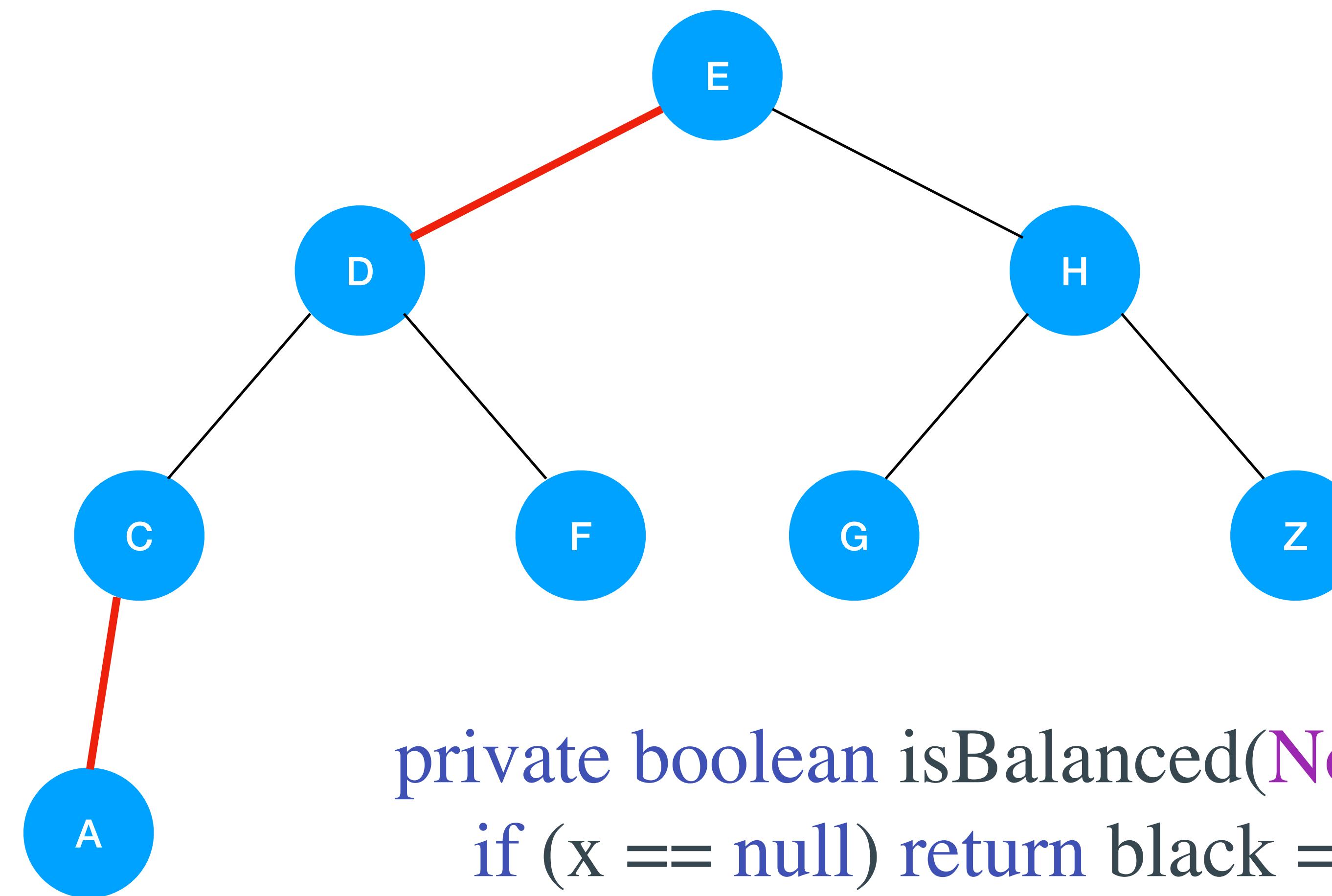
Question 3.1.6 IS RED BLACK



Node = A
Black = 0

```
private boolean isBalanced(Node x, int black) {  
    if (x == null) return black == 0;  
    if (!isRed(x)) black--;  
    return isBalanced(x.left, black) && isBalanced(x.right, black);  
}
```

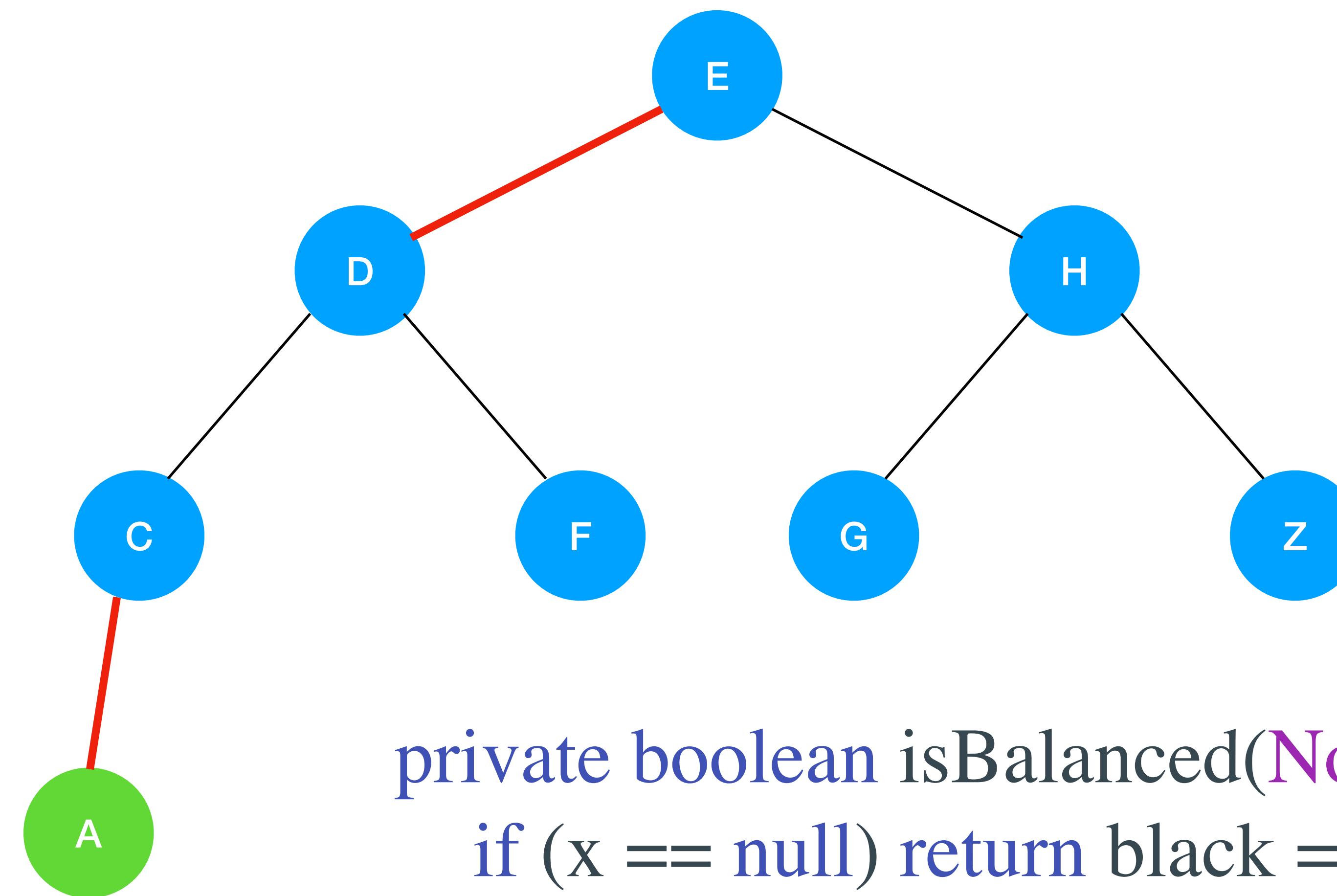
Question 3.1.6 IS RED BLACK



Node = null
Black = 0

```
private boolean isBalanced(Node x, int black) {  
    if (x == null) return black == 0;  
    if (!isRed(x)) black--;  
    return isBalanced(x.left, black) && isBalanced(x.right, black);  
}
```

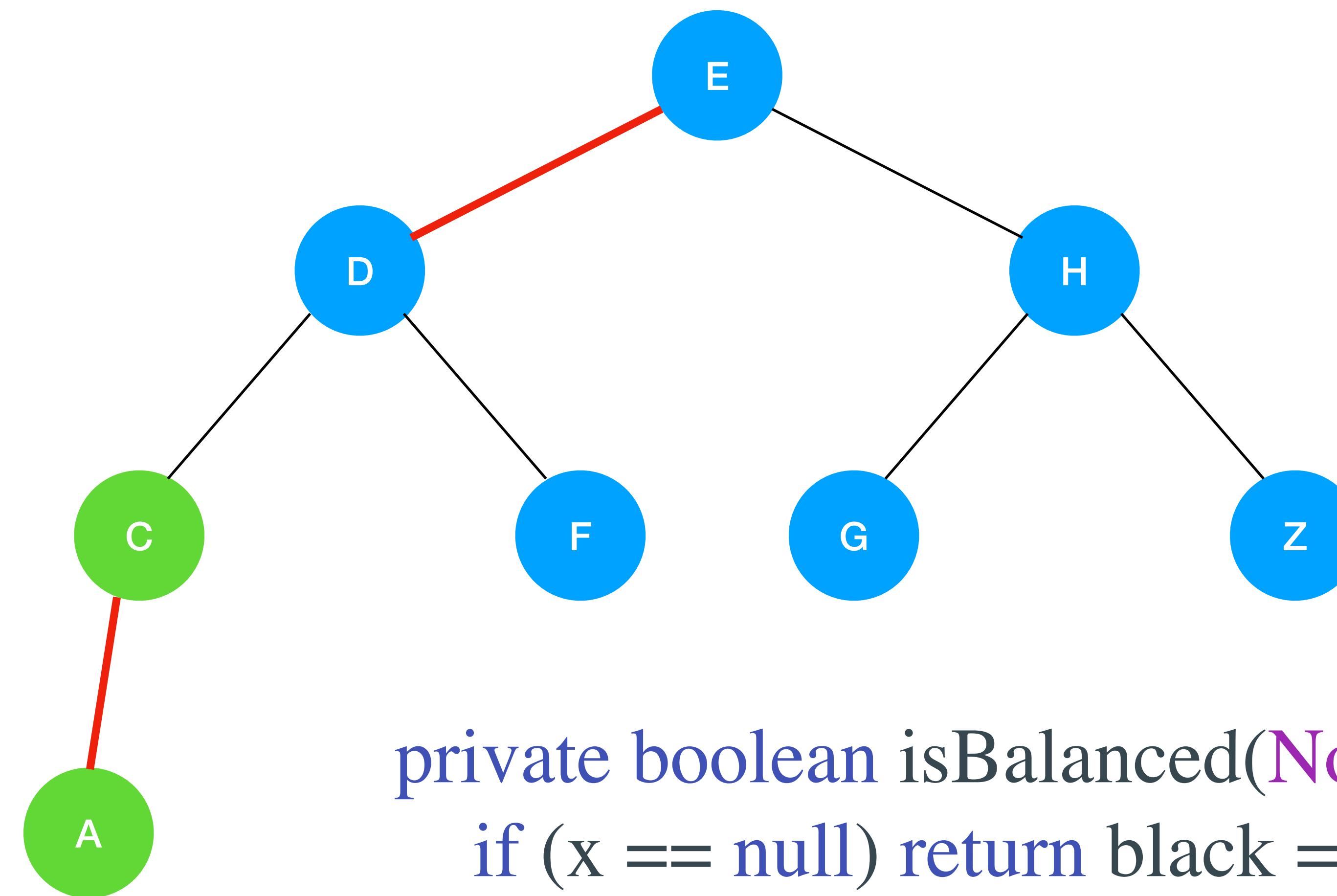
Question 3.1.6 IS RED BLACK



Node = A
Black = 0

```
private boolean isBalanced(Node x, int black) {  
    if (x == null) return black == 0;  
    if (!isRed(x)) black--;  
    return isBalanced(x.left, black) && isBalanced(x.right, black);  
}
```

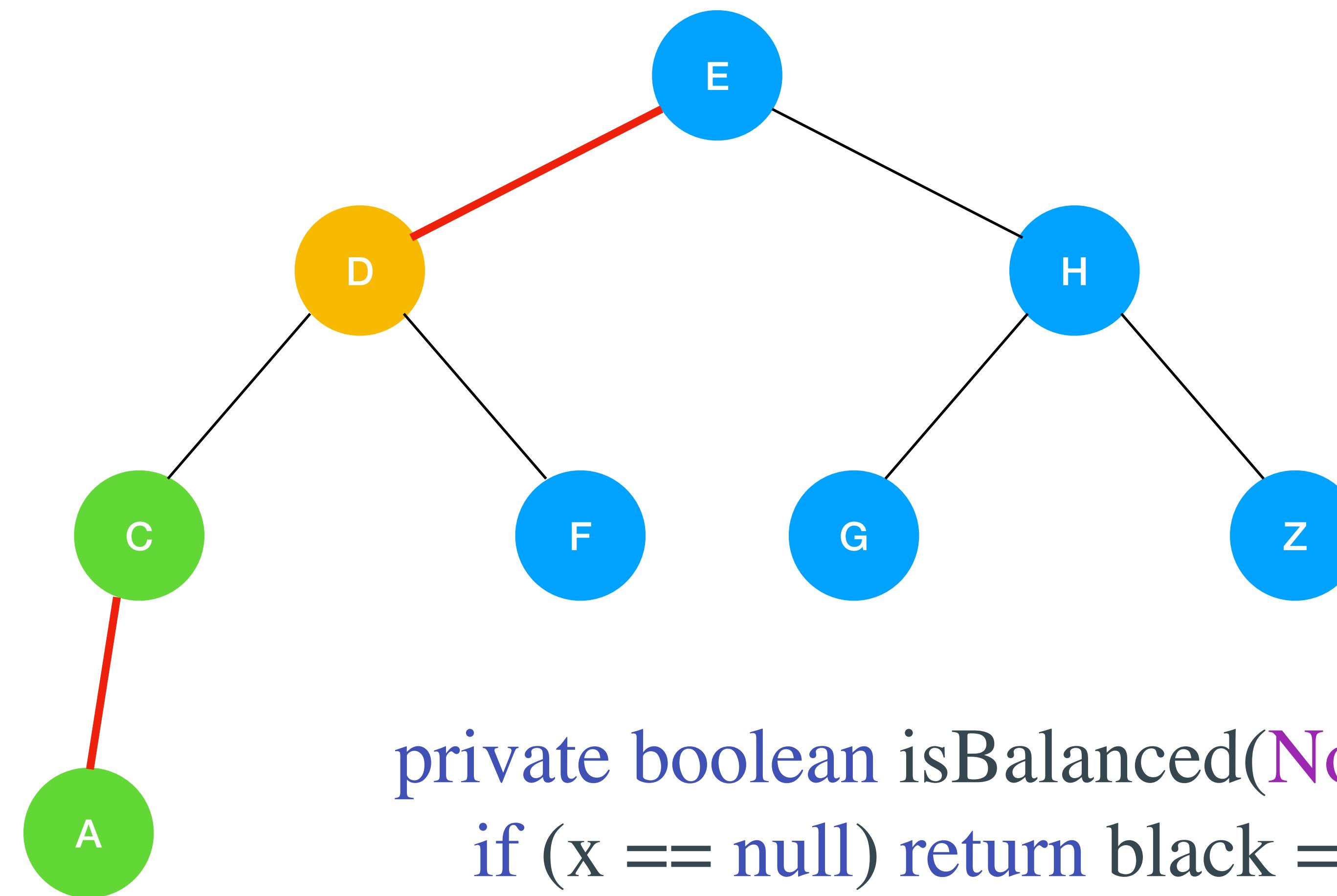
Question 3.1.6 IS RED BLACK



Node = C
Black = 0

```
private boolean isBalanced(Node x, int black) {  
    if (x == null) return black == 0;  
    if (!isRed(x)) black--;  
    return isBalanced(x.left, black) && isBalanced(x.right, black);  
}
```

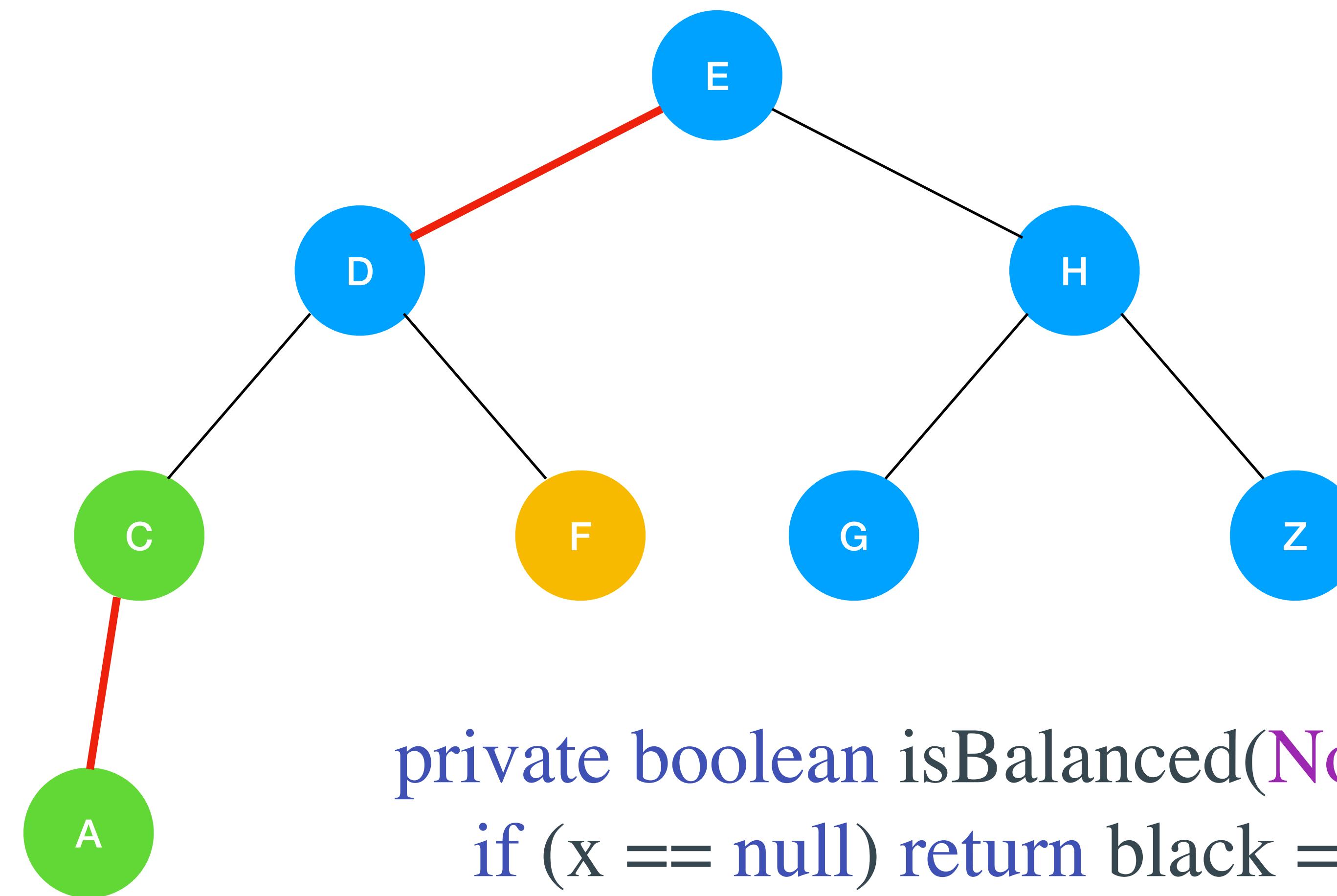
Question 3.1.6 IS RED BLACK



Node = D
Black = 1

```
private boolean isBalanced(Node x, int black) {  
    if (x == null) return black == 0;  
    if (!isRed(x)) black--;  
    return isBalanced(x.left, black) && isBalanced(x.right, black);  
}
```

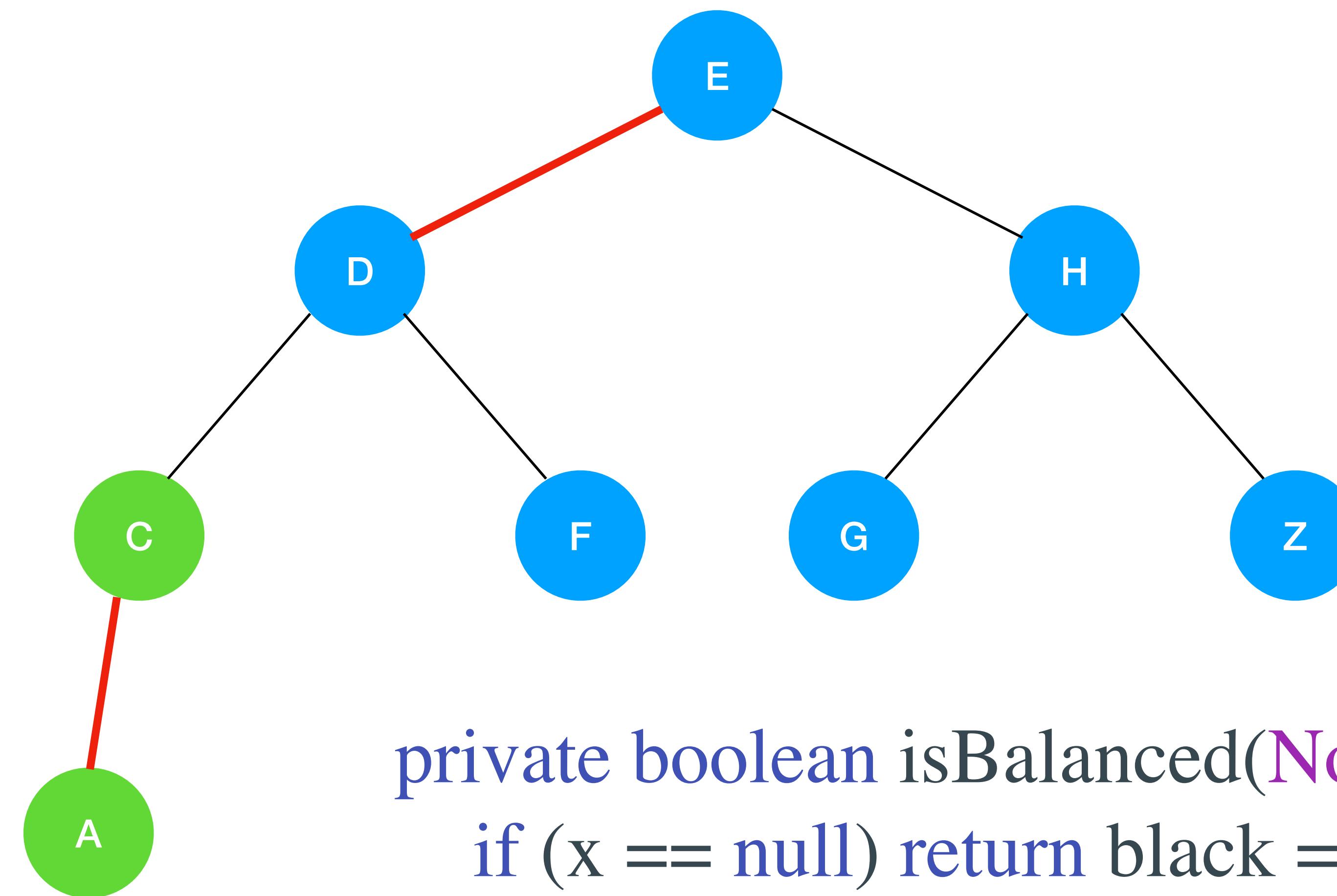
Question 3.1.6 IS RED BLACK



Node = F
Black = 0

```
private boolean isBalanced(Node x, int black) {  
    if (x == null) return black == 0;  
    if (!isRed(x)) black--;  
    return isBalanced(x.left, black) && isBalanced(x.right, black);  
}
```

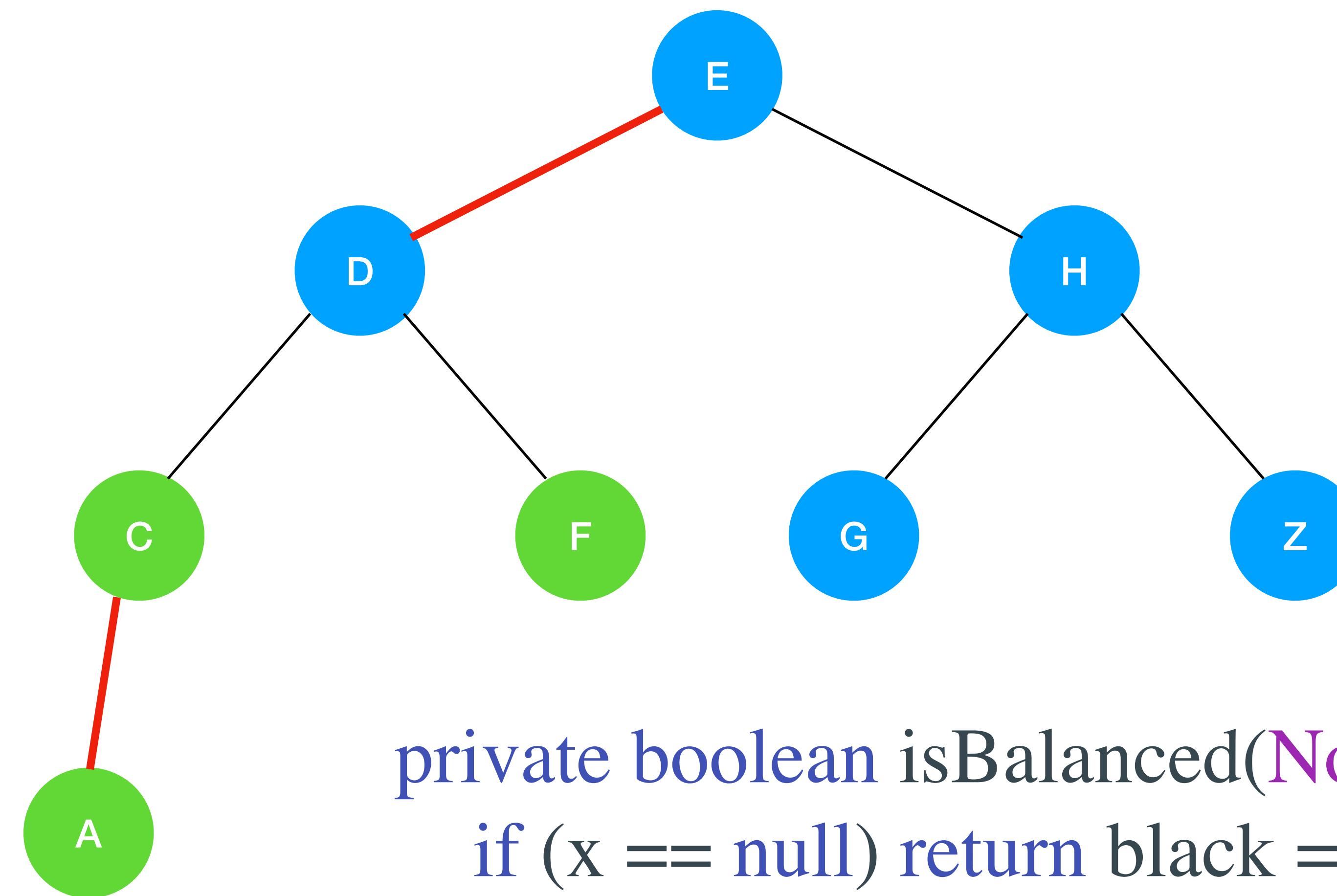
Question 3.1.6 IS RED BLACK



Node = null
Black = 0

```
private boolean isBalanced(Node x, int black) {  
    if (x == null) return black == 0;  
    if (!isRed(x)) black--;  
    return isBalanced(x.left, black) && isBalanced(x.right, black);  
}
```

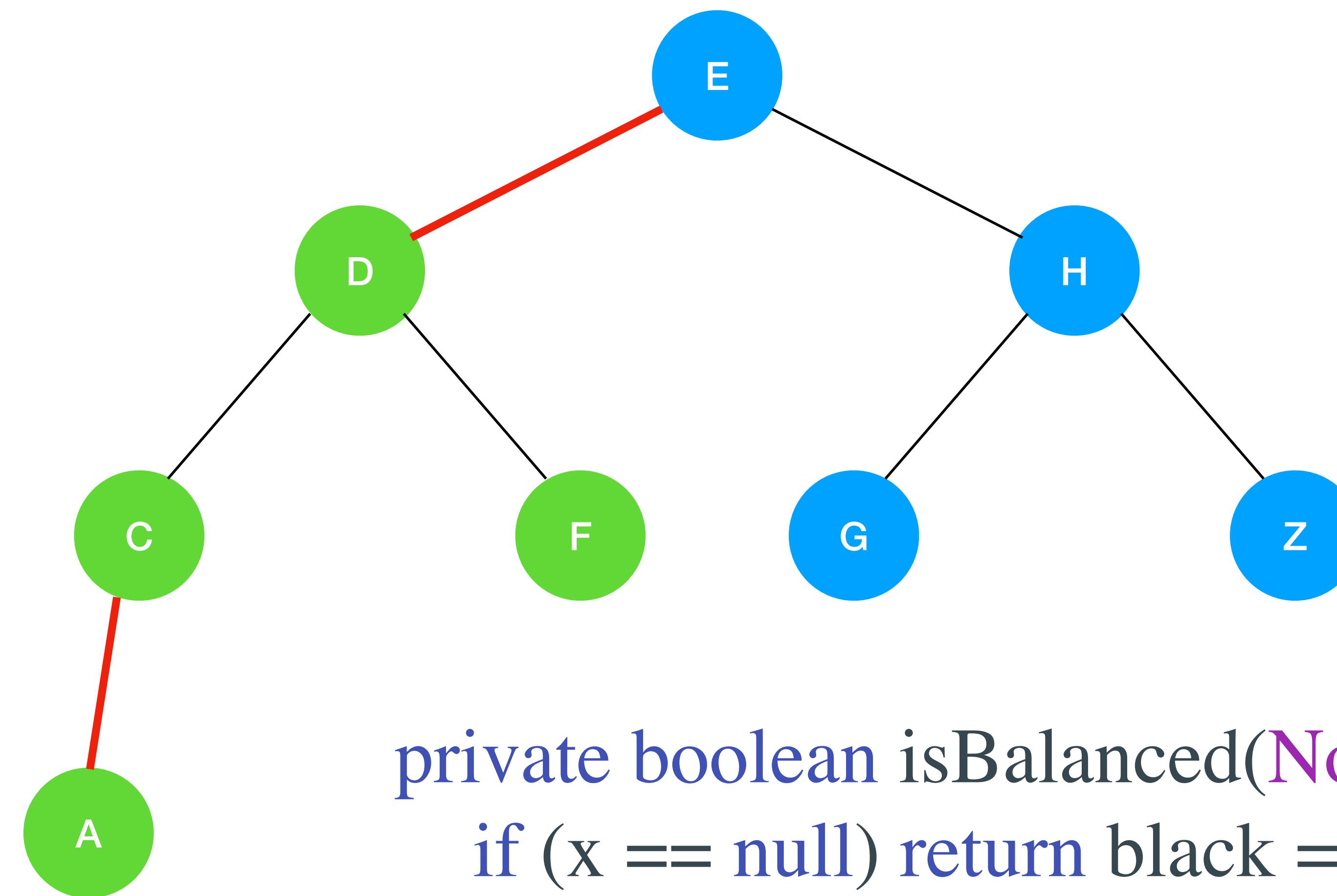
Question 3.1.6 IS RED BLACK



Node = F
Black = 0

```
private boolean isBalanced(Node x, int black) {  
    if (x == null) return black == 0;  
    if (!isRed(x)) black--;  
    return isBalanced(x.left, black) && isBalanced(x.right, black);  
}
```

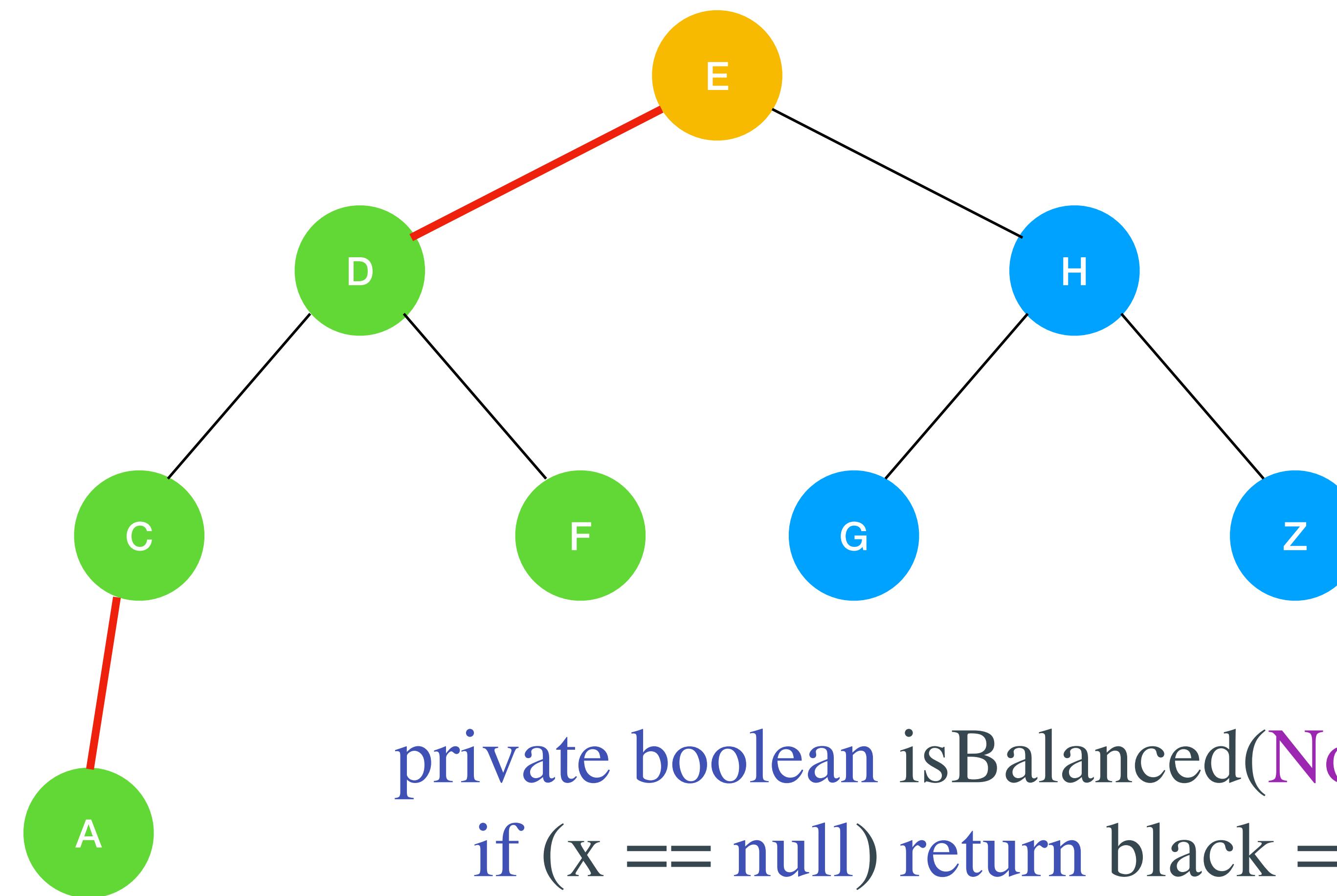
Question 3.1.6 IS RED BLACK



Node = D
Black = 1

```
private boolean isBalanced(Node x, int black) {  
    if (x == null) return black == 0;  
    if (!isRed(x)) black--;  
    return isBalanced(x.left, black) && isBalanced(x.right, black);  
}
```

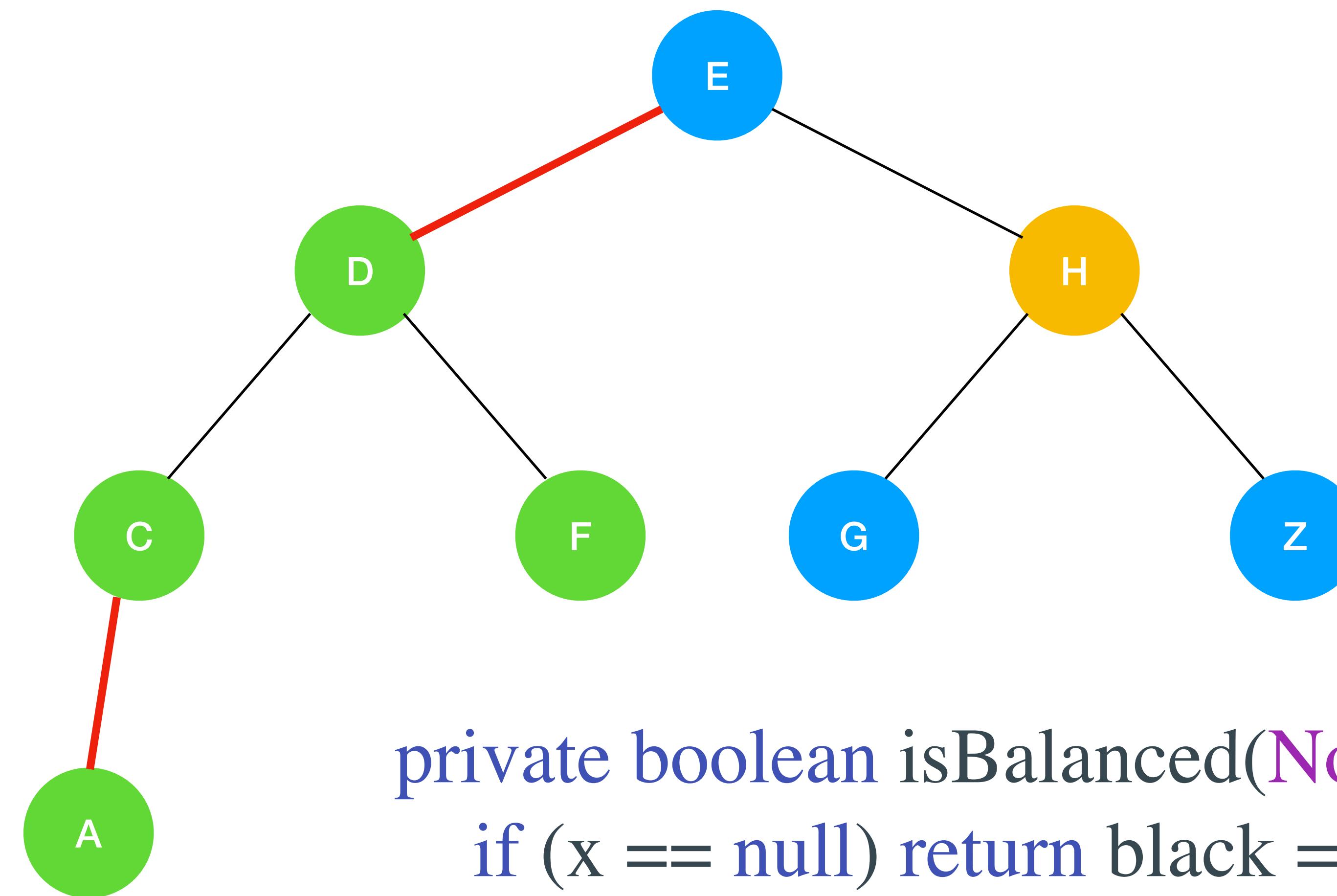
Question 3.1.6 IS RED BLACK



Node = E
Black = 1

```
private boolean isBalanced(Node x, int black) {  
    if (x == null) return black == 0;  
    if (!isRed(x)) black--;  
    return isBalanced(x.left, black) && isBalanced(x.right, black);  
}
```

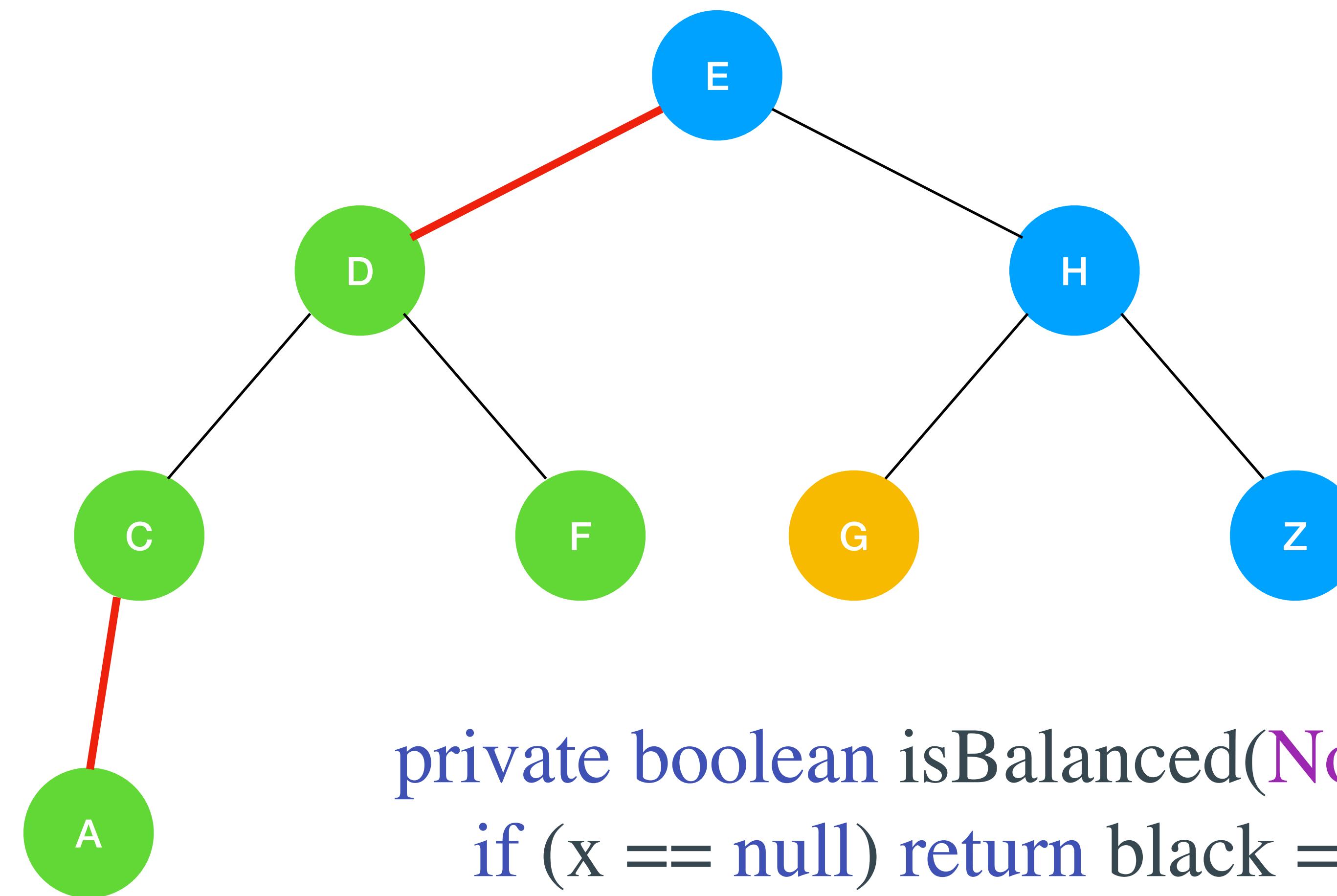
Question 3.1.6 IS RED BLACK



Node = H
Black = 0

```
private boolean isBalanced(Node x, int black) {  
    if (x == null) return black == 0;  
    if (!isRed(x)) black--;  
    return isBalanced(x.left, black) && isBalanced(x.right, black);  
}
```

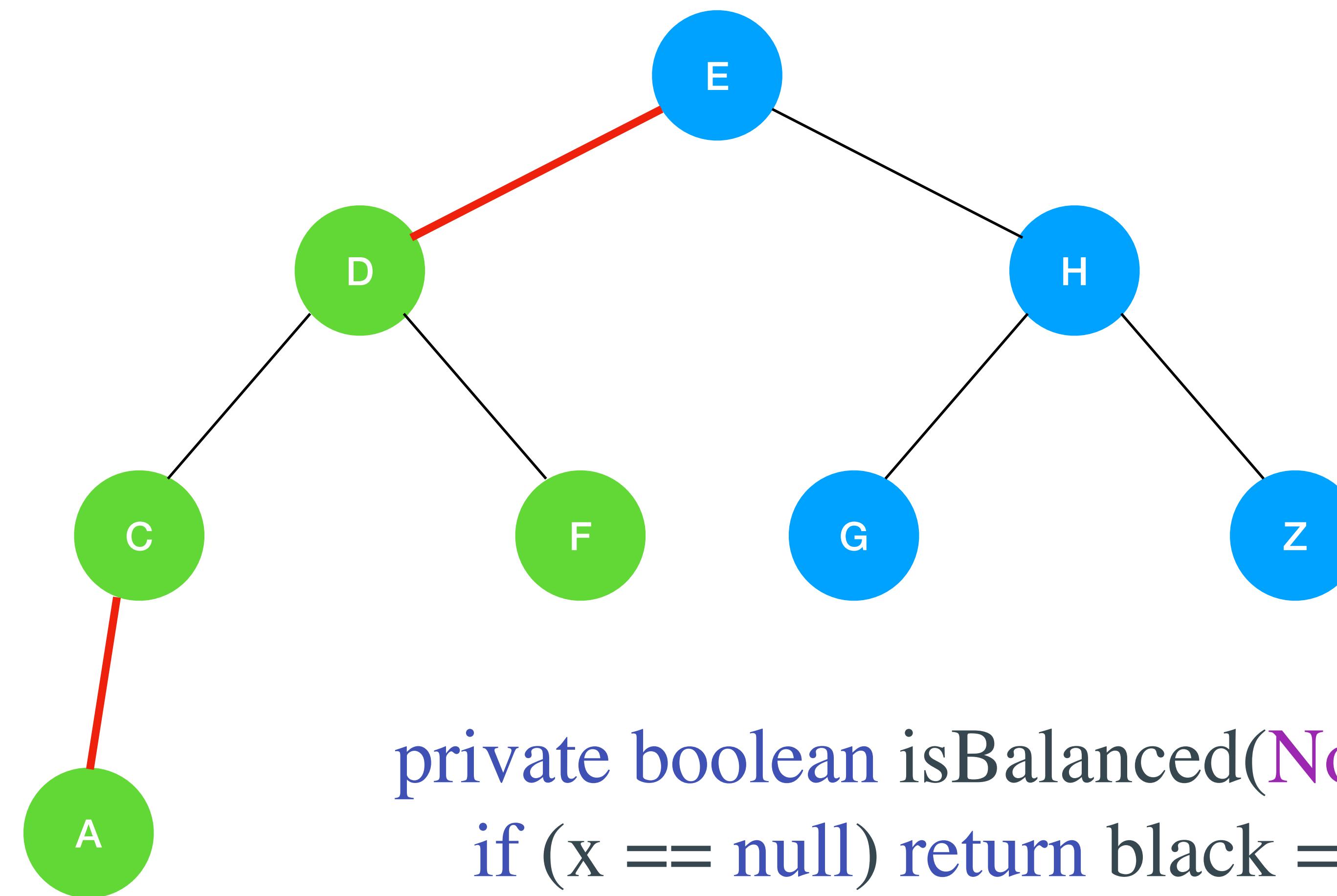
Question 3.1.6 IS RED BLACK



Node = G
Black = -1

```
private boolean isBalanced(Node x, int black) {  
    if (x == null) return black == 0;  
    if (!isRed(x)) black--;  
    return isBalanced(x.left, black) && isBalanced(x.right, black);  
}
```

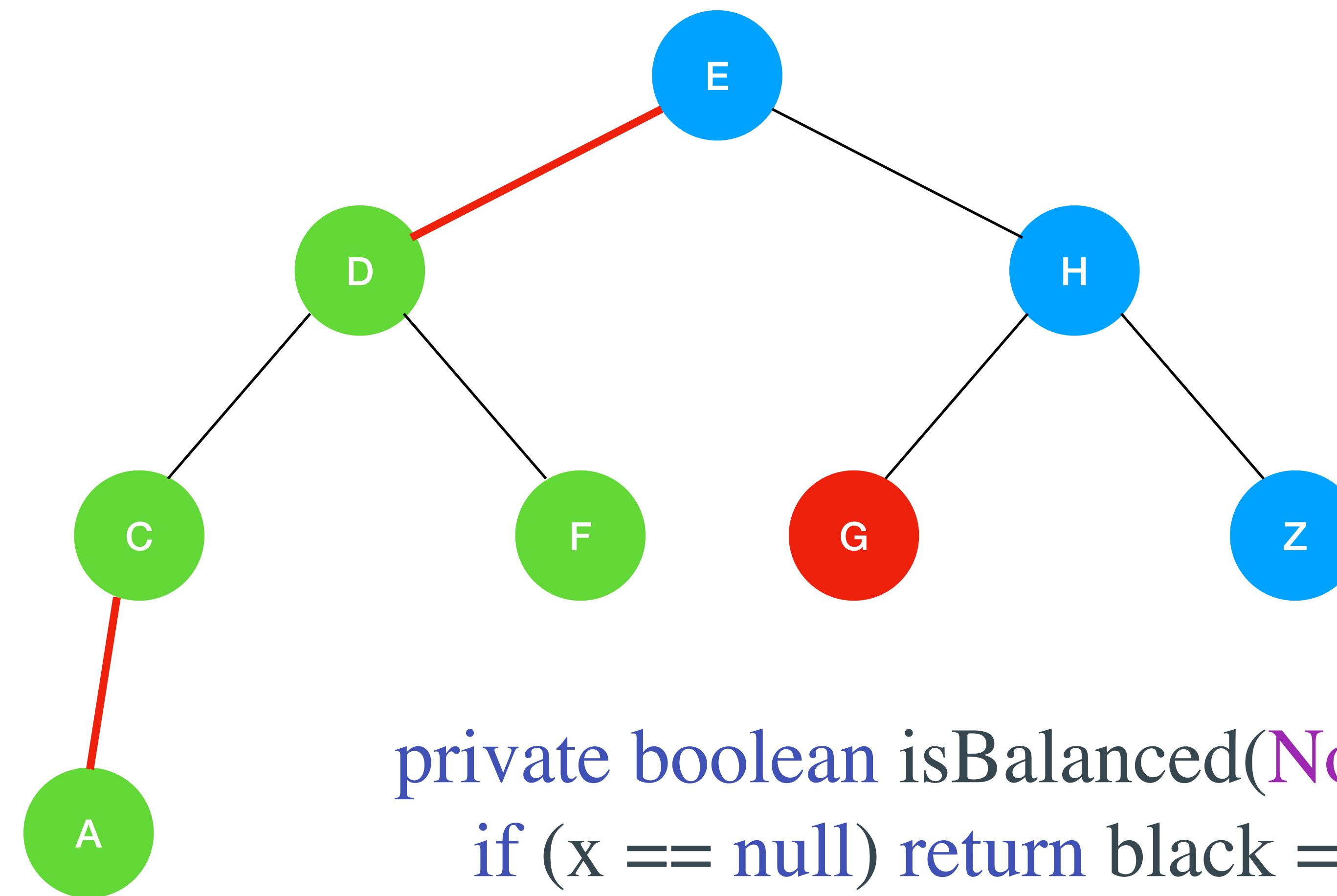
Question 3.1.6 IS RED BLACK



Node = null
Black = -1

```
private boolean isBalanced(Node x, int black) {  
    if (x == null) return black == 0;  
    if (!isRed(x)) black--;  
    return isBalanced(x.left, black) && isBalanced(x.right, black);  
}
```

Question 3.1.6 IS RED BLACK



Node = G
Black = -1

```
private boolean isBalanced(Node x, int black) {  
    if (x == null) return black == 0;  
    if (!isRed(x)) black--;  
    return isBalanced(x.left, black) && isBalanced(x.right, black);  
}
```

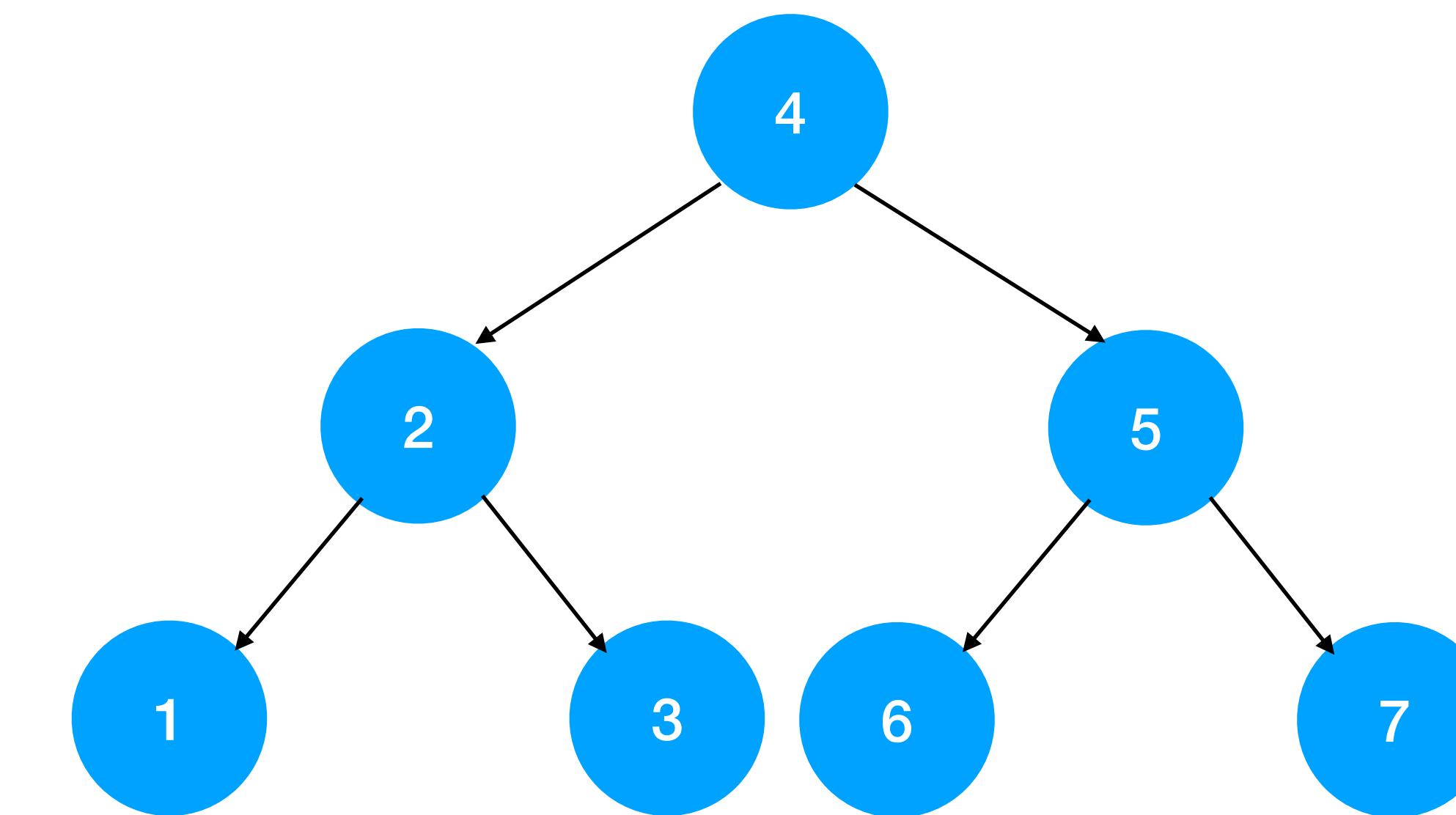
algorithm (data structure)	worst-case cost (after N inserts)		average-case cost (after N random inserts)		efficiently support ordered operations?
	search	insert	search hit	insert	
<i>sequential search (unordered linked list)</i>	N	N	N/2	N	no
<i>binary search (ordered array)</i>	$\lg N$	N	$\lg N$	N/2	yes
<i>binary tree search (BST)</i>	N	N	$1.39 \lg N$	$1.39 \lg N$	yes
<i>2-3 tree search (red-black BST)</i>	$2 \lg N$	$2 \lg N$	$1.00 \lg N$	$1.00 \lg N$	yes

Cost summary for symbol-table implementations (updated)

Question 3.1.10 : Parcours préfixe

Parcours préfixe :

- 1. Traitement de la racine**
- 2. Traitement de la branche gauche**
- 3. Traitement de la branche droite**

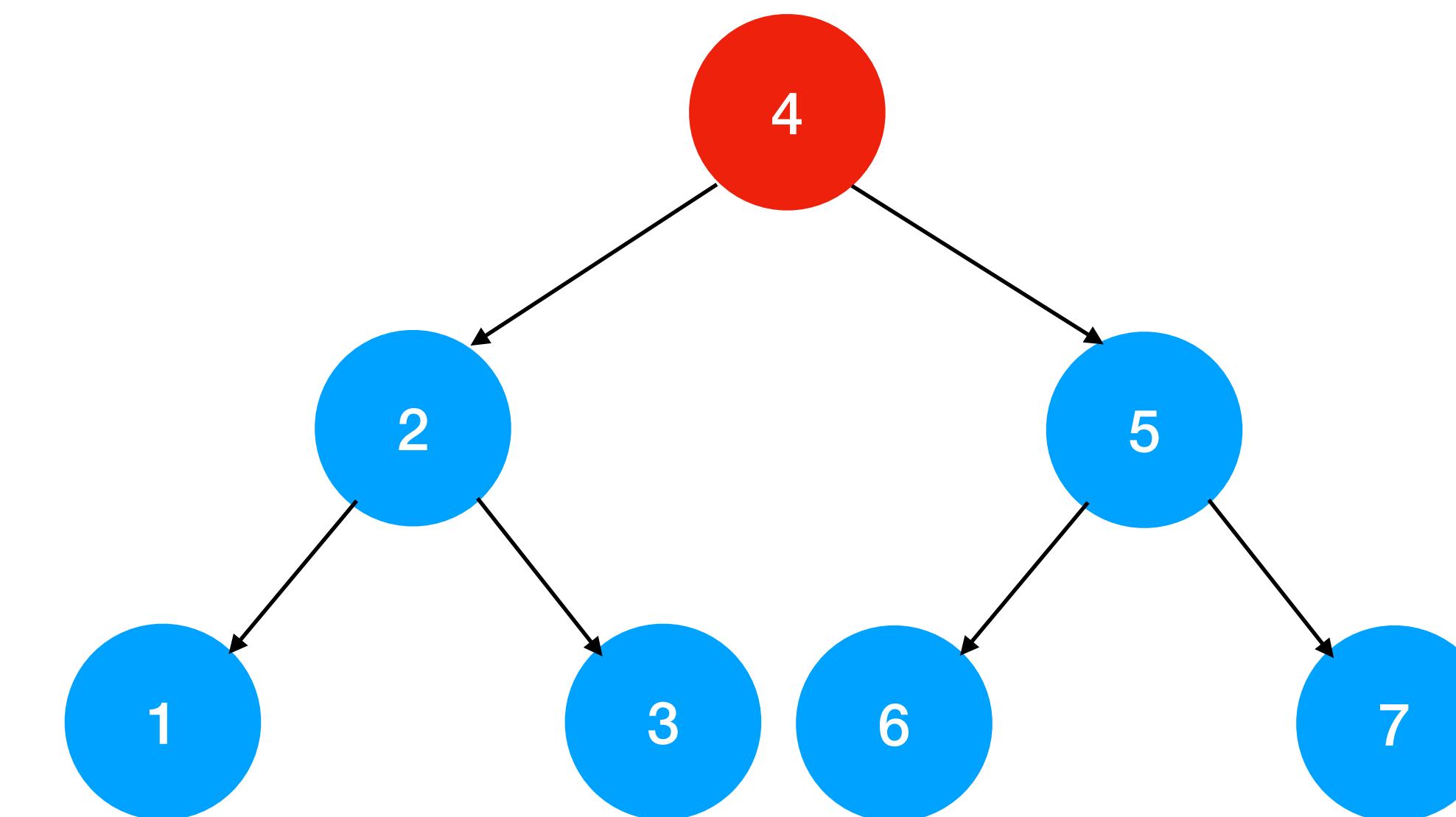


Question 3.1.10 : Parcours préfixe

Parcours préfixe :

- 1. Traitement de la racine**
- 2. Traitement de la branche gauche**
- 3. Traitement de la branche droite**

4

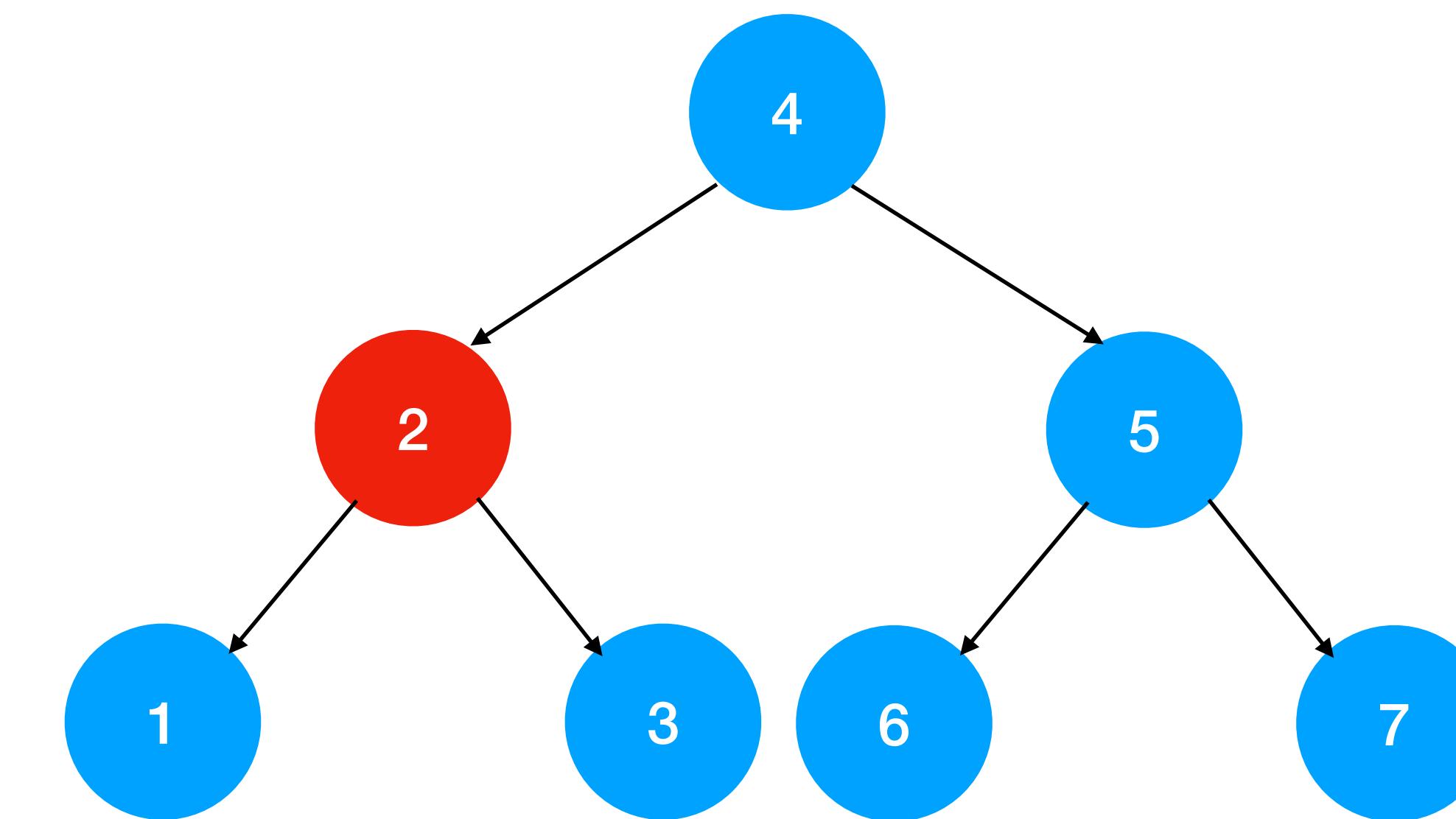


Question 3.1.10 : Parcours préfixe

Parcours préfixe :

- 1. Traitement de la racine**
- 2. Traitement de la branche gauche**
- 3. Traitement de la branche droite**

4, 2

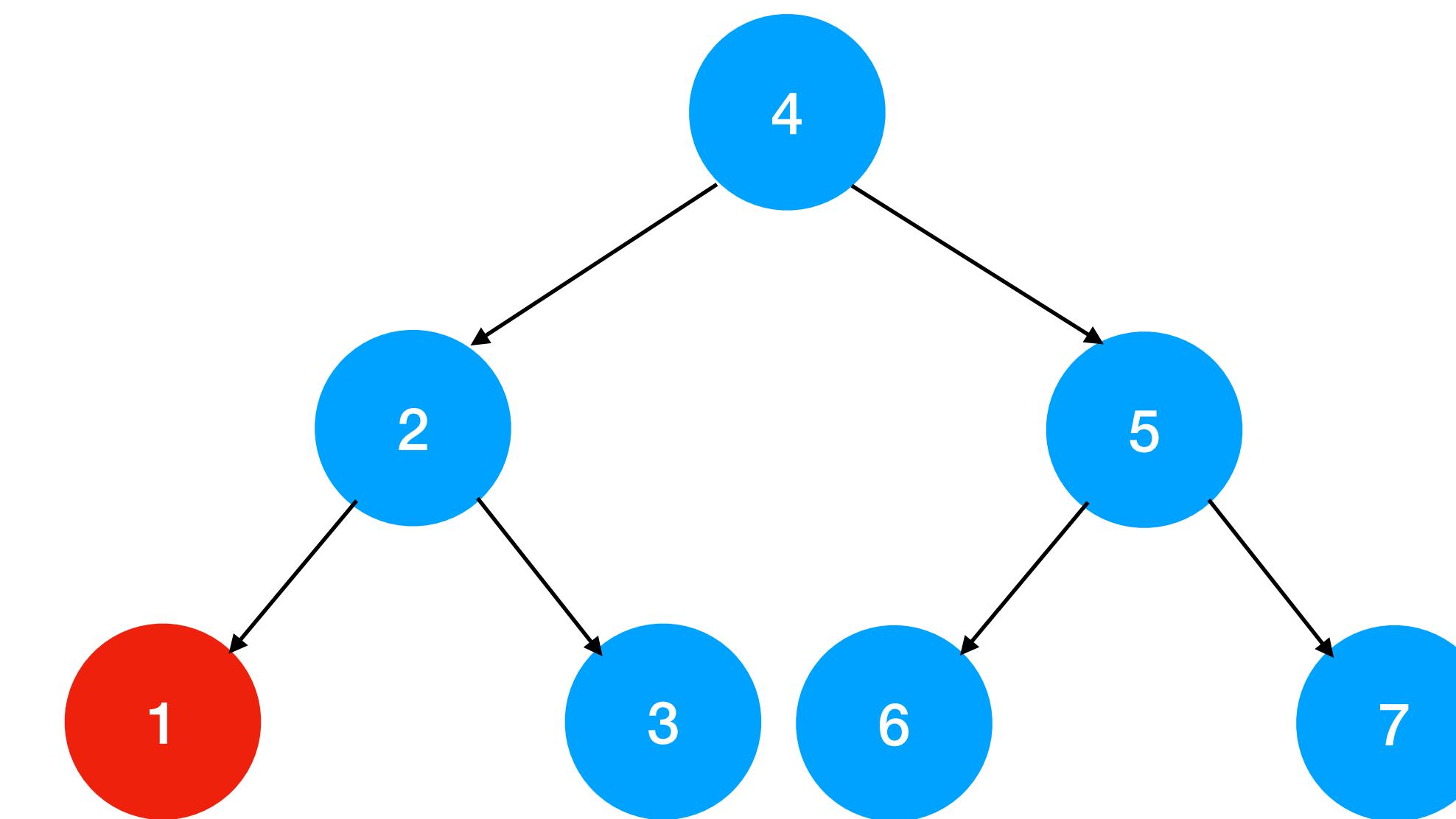


Question 3.1.10 : Parcours préfixe

Parcours préfixe :

- 1. Traitement de la racine**
- 2. Traitement de la branche gauche**
- 3. Traitement de la branche droite**

4, 2, 1

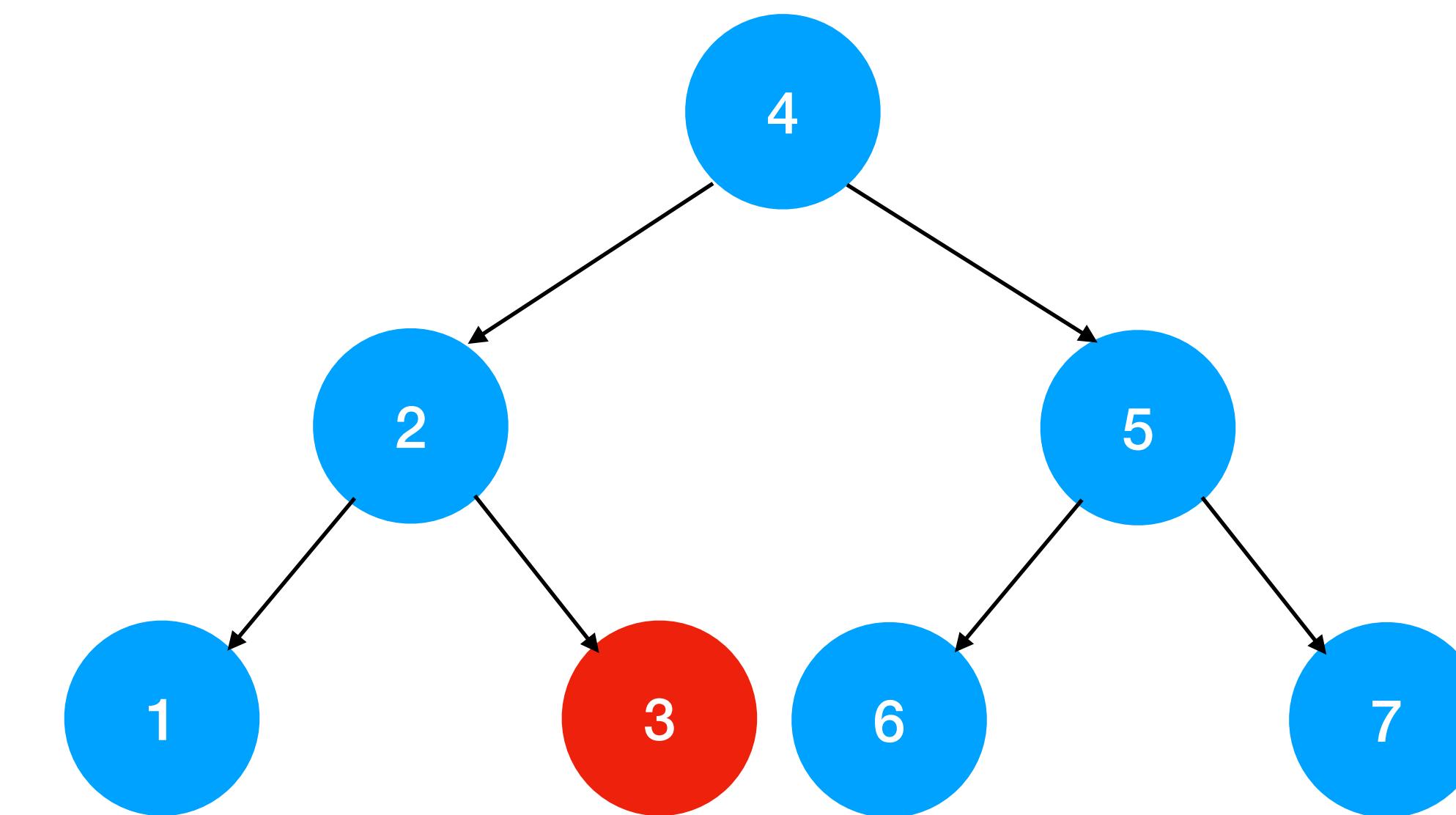


Question 3.1.10 : Parcours préfixe

Parcours préfixe :

- 1. Traitement de la racine**
- 2. Traitement de la branche gauche**
- 3. Traitement de la branche droite**

4, 2, 1, 3

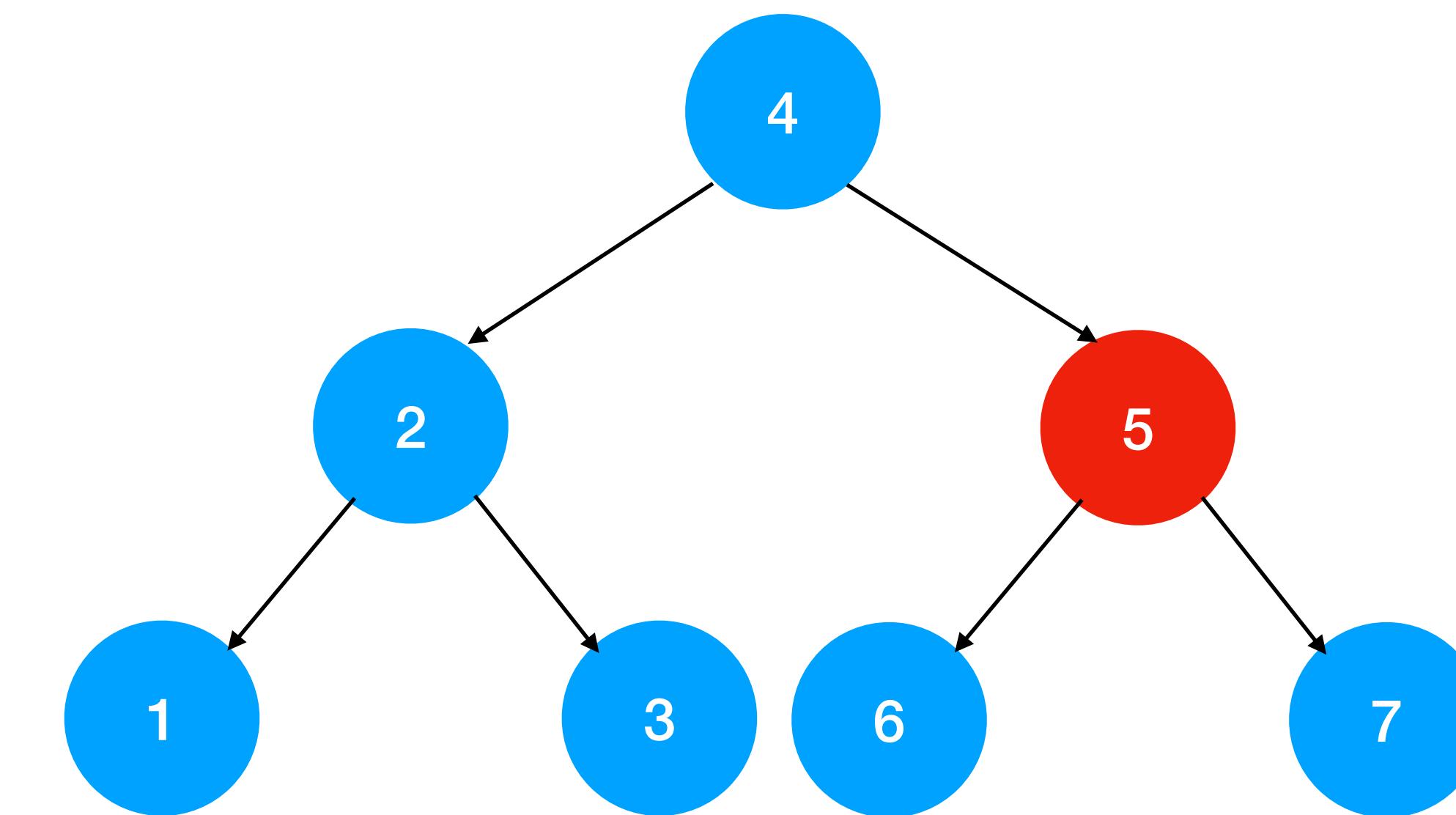


Question 3.1.10 : Parcours préfixe

Parcours préfixe :

- 1. Traitement de la racine**
- 2. Traitement de la branche gauche**
- 3. Traitement de la branche droite**

4, 2, 1, 3, 5

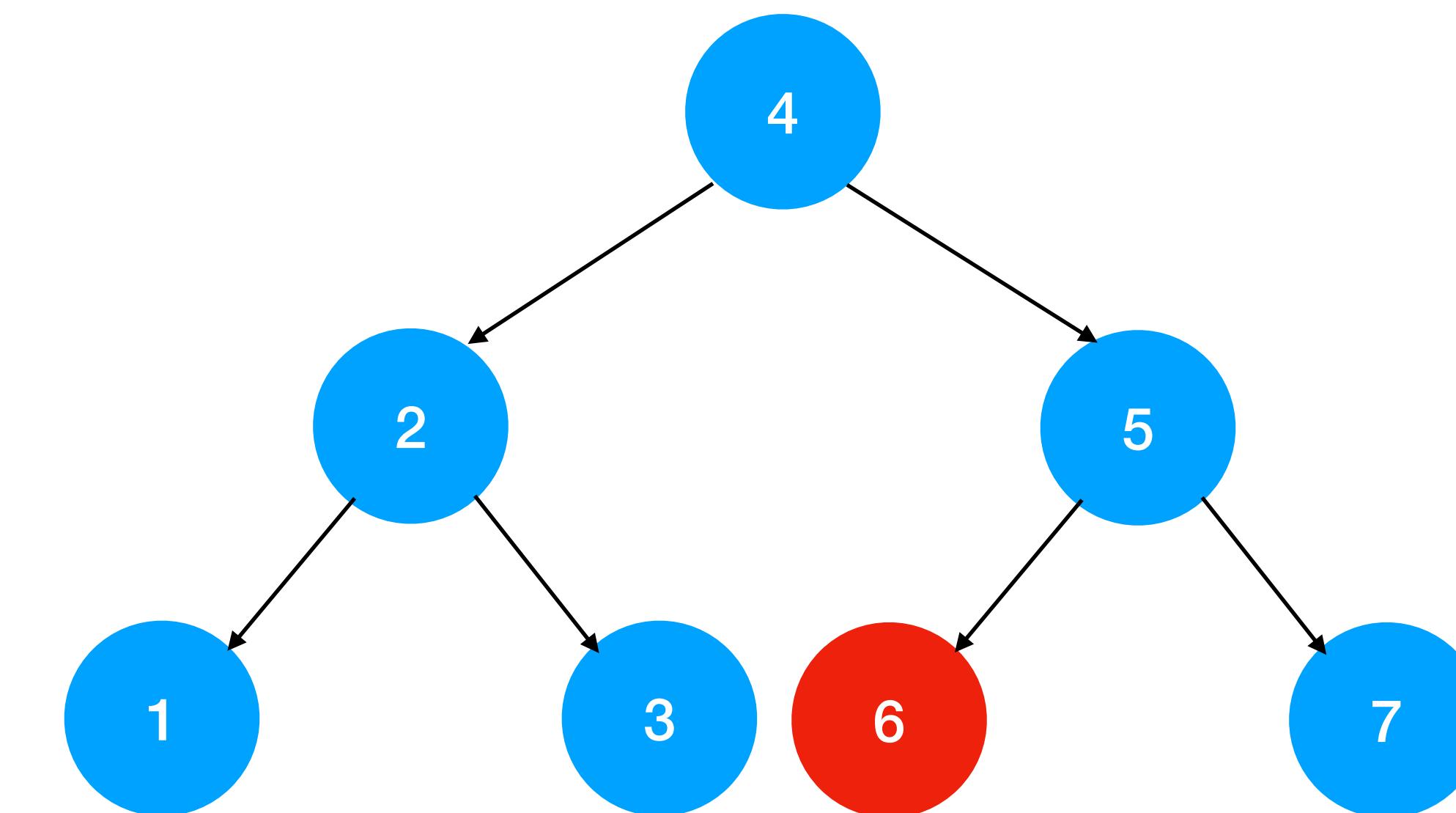


Question 3.1.10 : Parcours préfixe

Parcours préfixe :

- 1. Traitement de la racine**
- 2. Traitement de la branche gauche**
- 3. Traitement de la branche droite**

4, 2, 1, 3, 5, 6

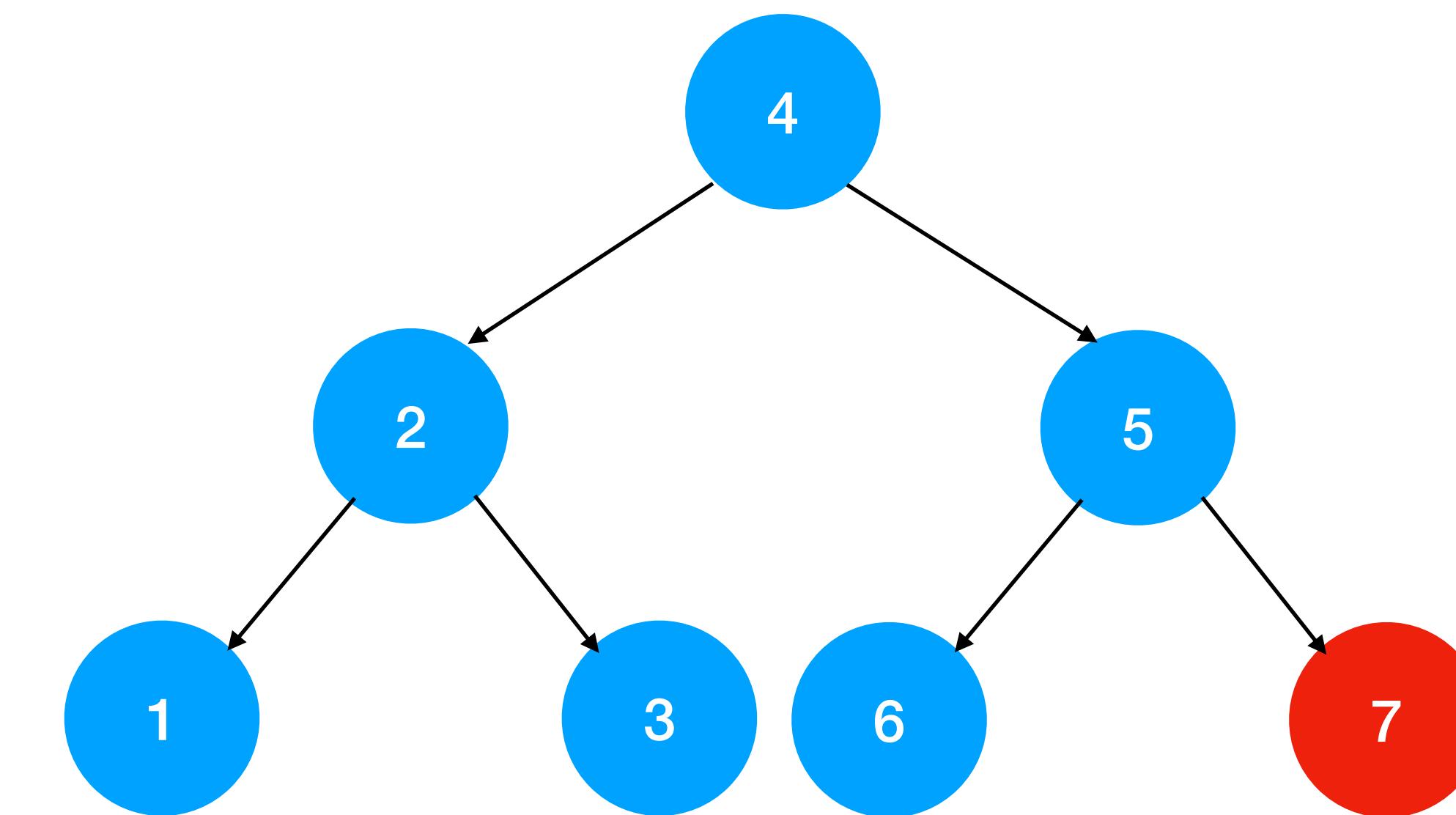


Question 3.1.10 : Parcours préfixe

Parcours préfixe :

- 1. Traitement de la racine**
- 2. Traitement de la branche gauche**
- 3. Traitement de la branche droite**

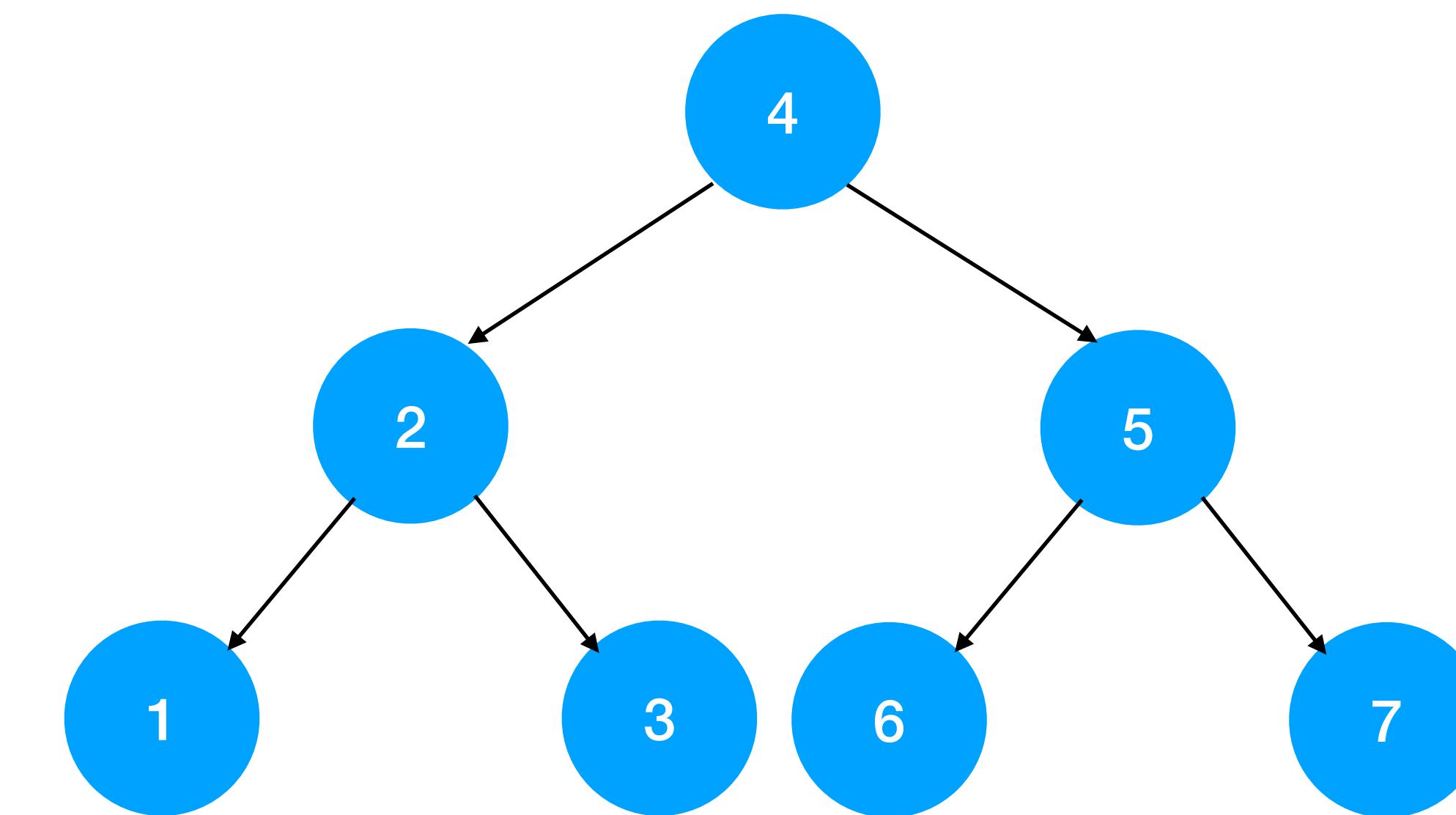
4, 2, 1, 3, 5, 6, 7



Question 3.1.10 : Parcours infixé

Parcours infixé :

- 1. Traitement de la branche gauche**
- 2. Traitement de la racine**
- 3. Traitement de la branche droite**

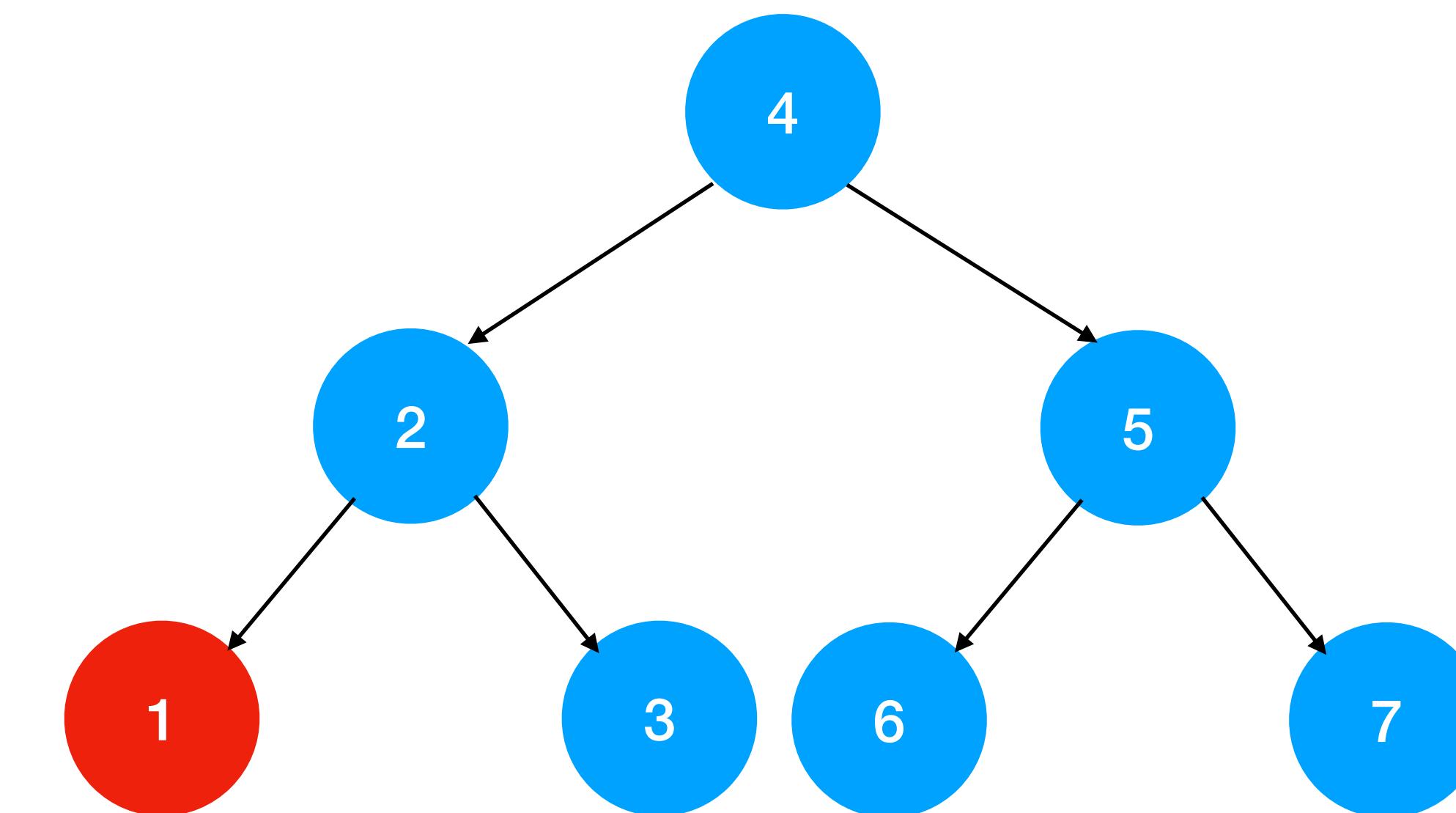


Question 3.1.10 : Parcours infixé

Parcours infixé :

- 1. Traitement de la branche gauche**
- 2. Traitement de la racine**
- 3. Traitement de la branche droite**

1

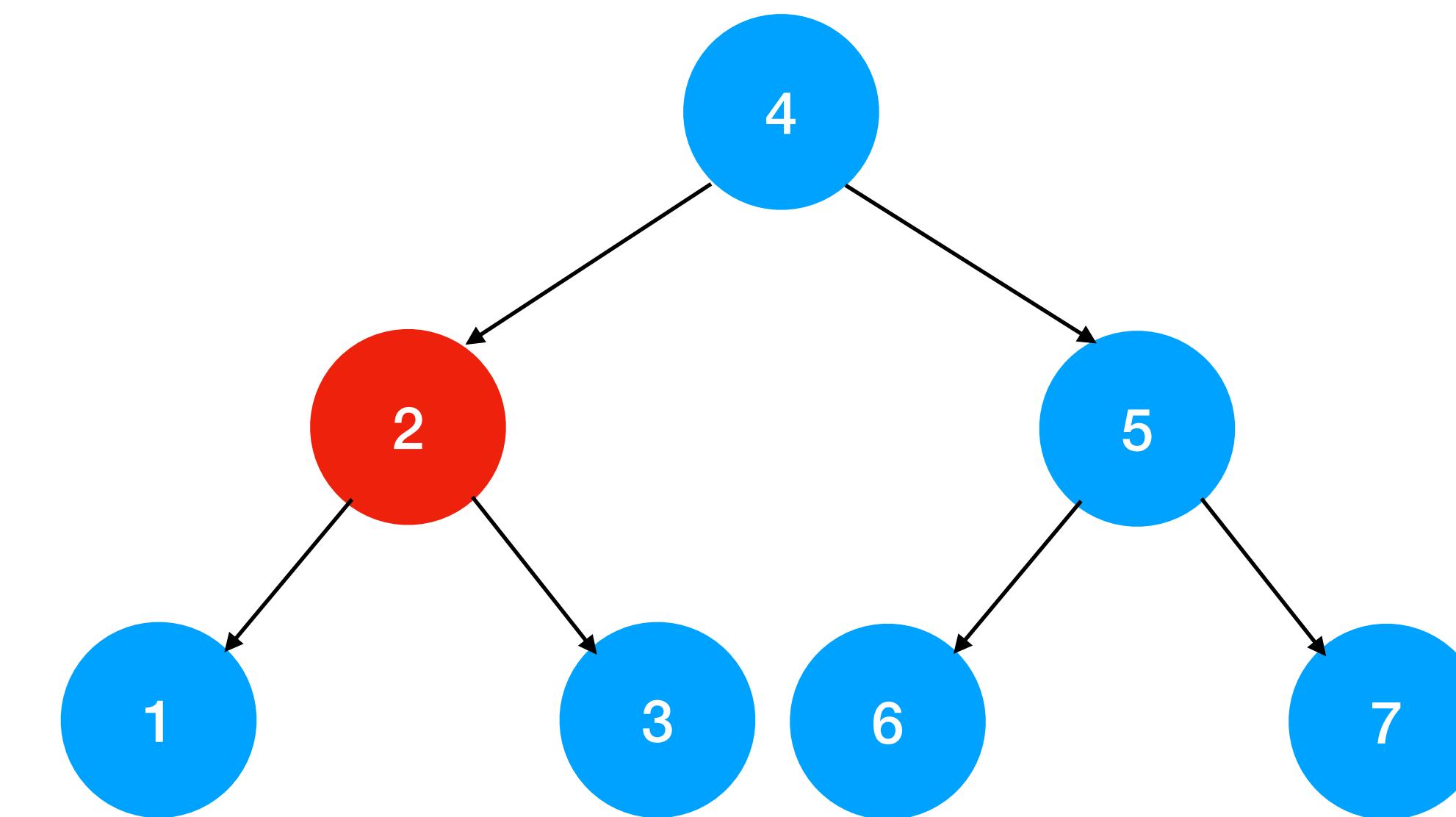


Question 3.1.10 : Parcours infixé

Parcours infixé :

1. Traitement de la branche gauche
2. Traitement de la racine
3. Traitement de la branche droite

1, 2

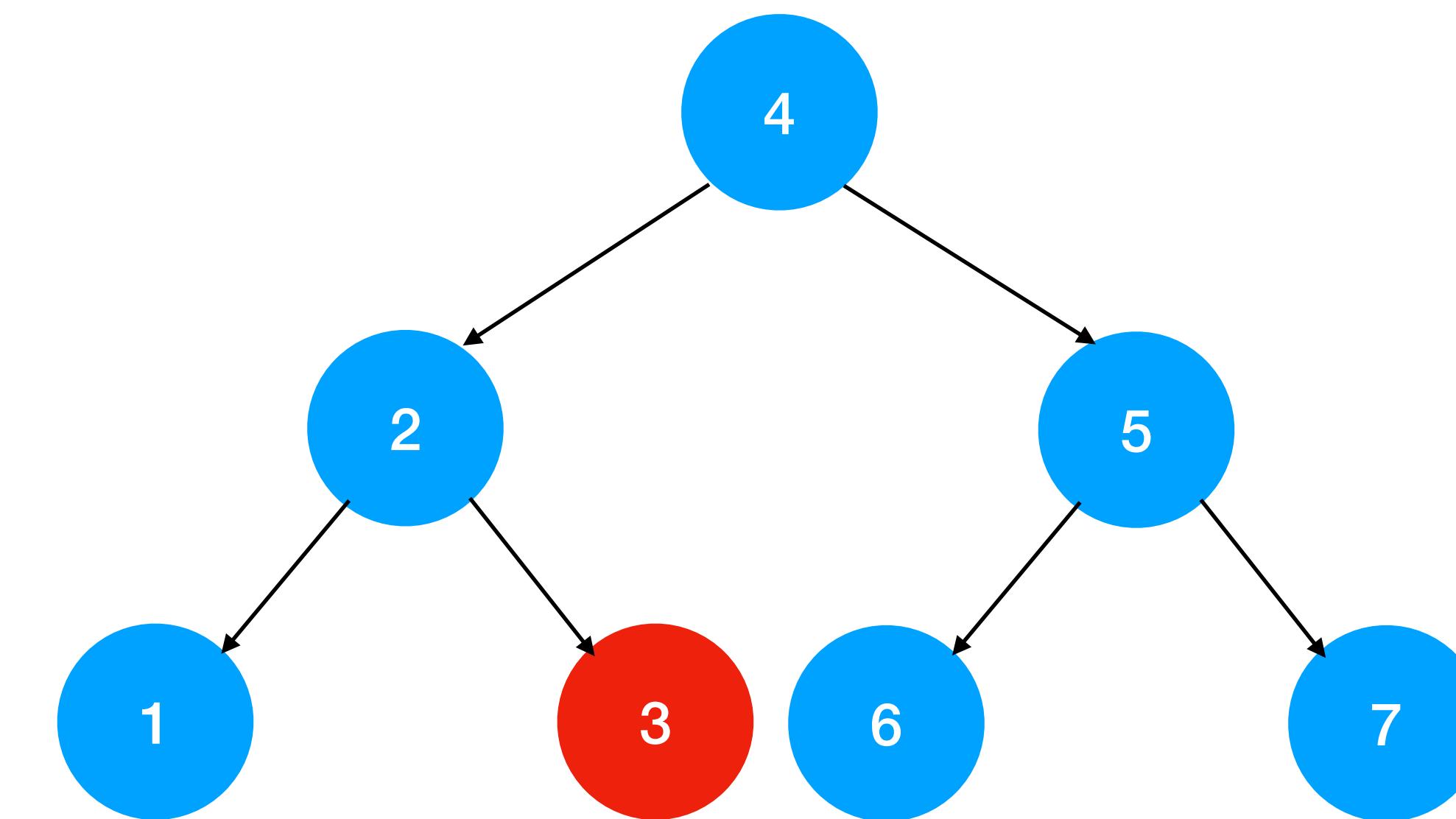


Question 3.1.10 : Parcours infixé

Parcours infixé :

- 1. Traitement de la branche gauche**
- 2. Traitement de la racine**
- 3. Traitement de la branche droite**

1, 2, 3

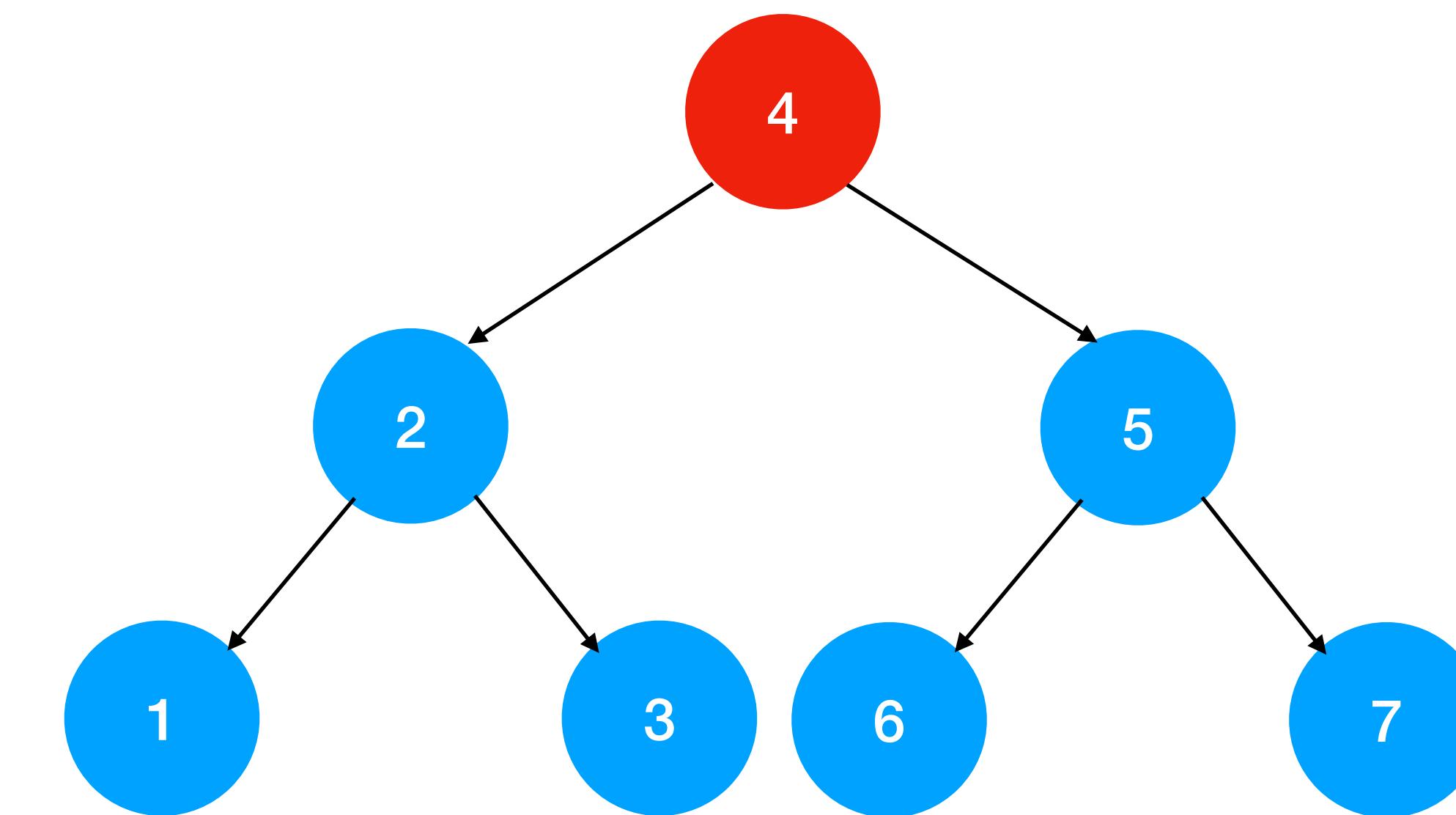


Question 3.1.10 : Parcours infixé

Parcours infixé :

- 1. Traitement de la branche gauche**
- 2. Traitement de la racine**
- 3. Traitement de la branche droite**

1, 2, 3, 4

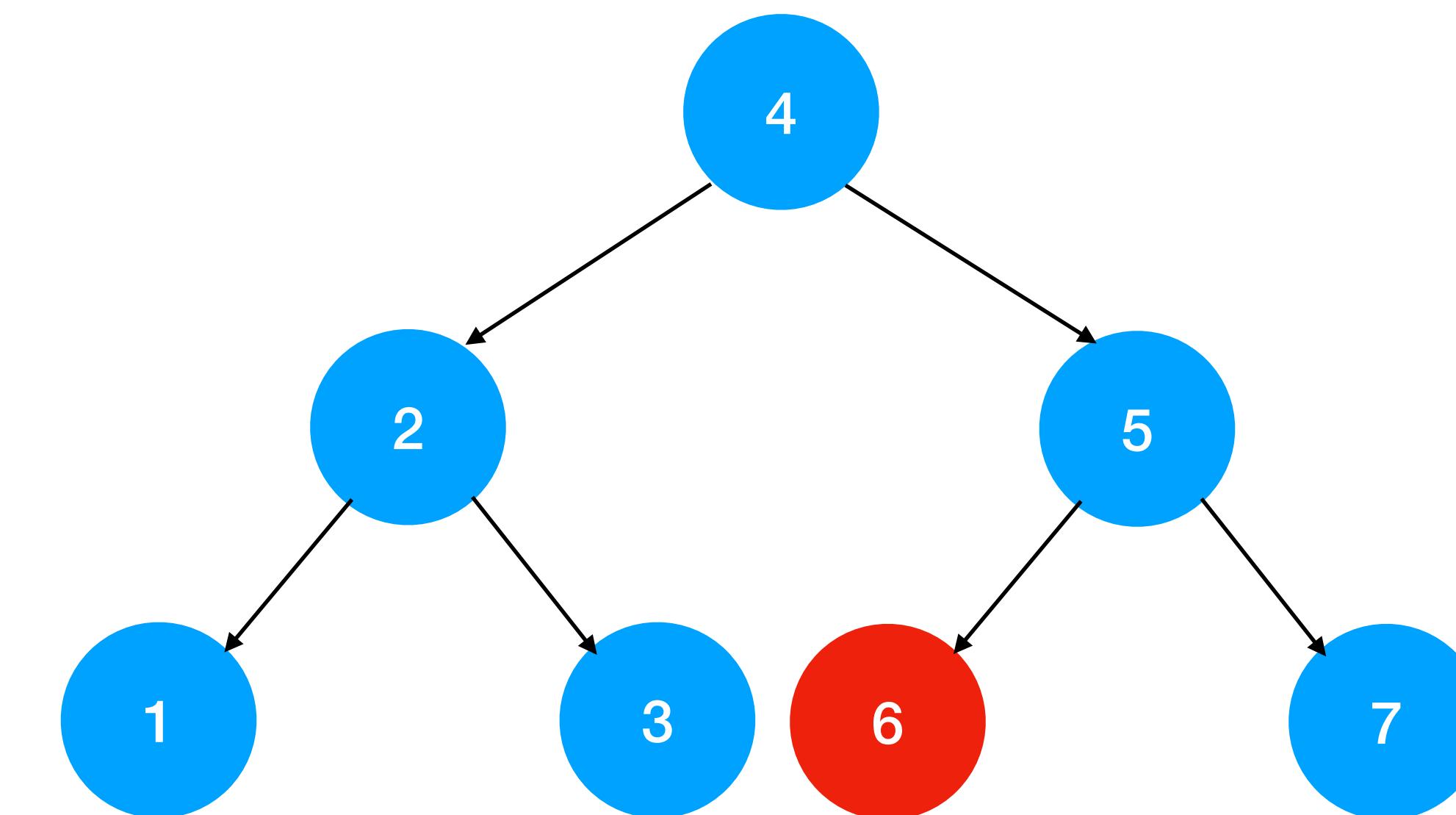


Question 3.1.10 : Parcours infixé

Parcours infixé :

1. Traitement de la branche gauche
2. Traitement de la racine
3. Traitement de la branche droite

1, 2, 3, 4, 6

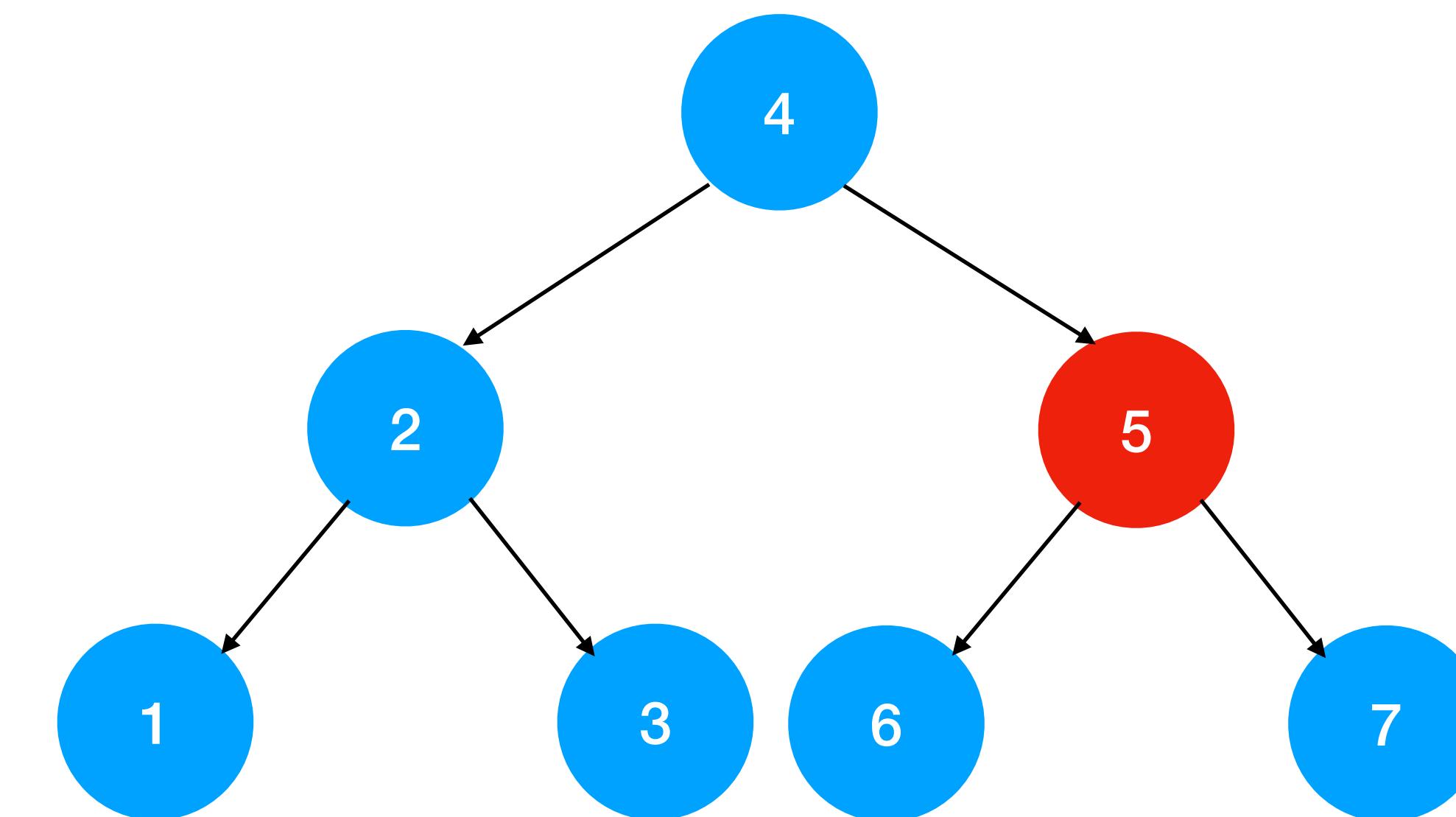


Question 3.1.10 : Parcours infixé

Parcours infixé :

1. Traitement de la branche gauche
2. Traitement de la racine
3. Traitement de la branche droite

1, 2, 3, 4, 6, 5

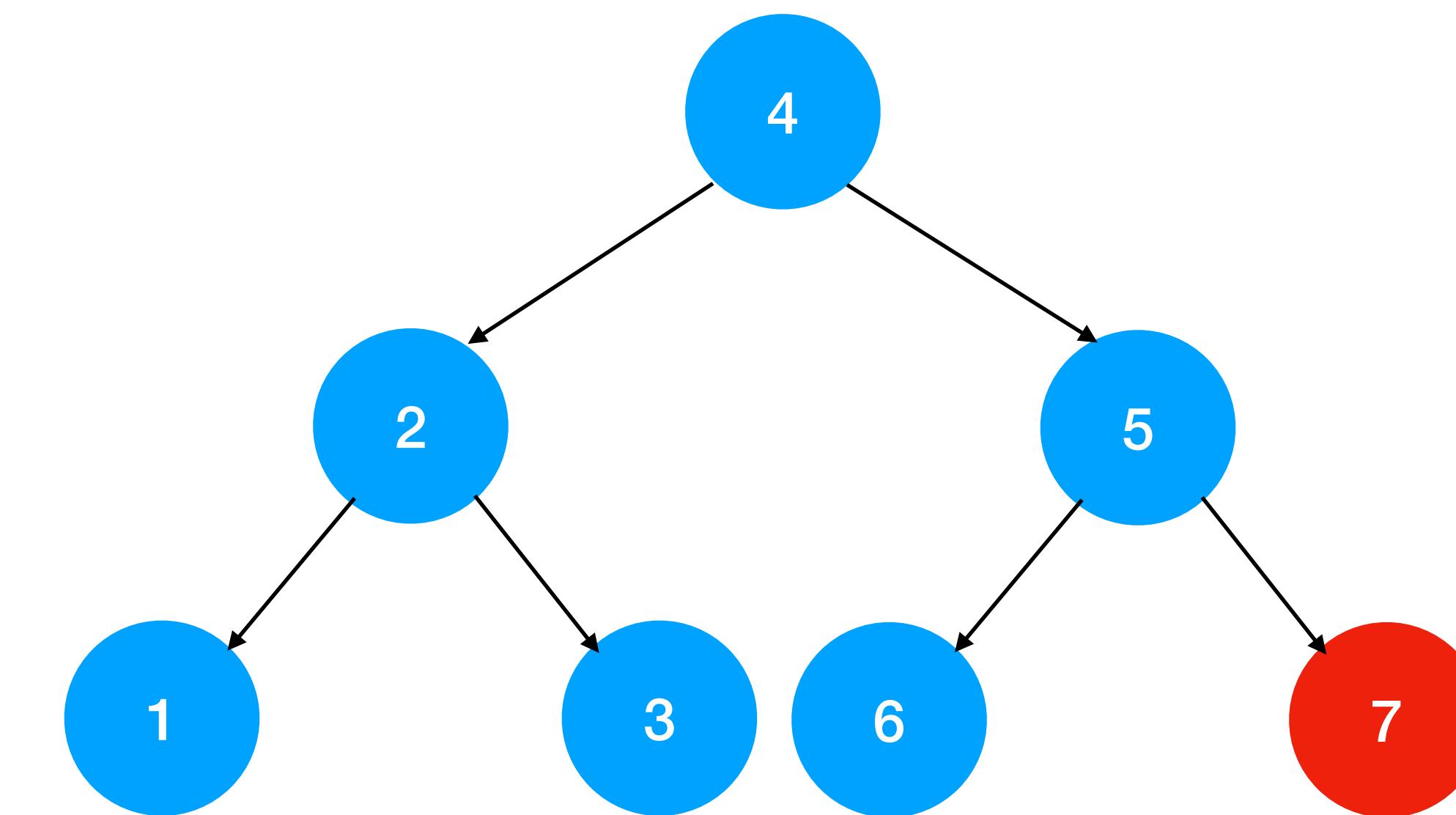


Question 3.1.10 : Parcours infixé

Parcours infixé :

1. Traitement de la branche gauche
2. Traitement de la racine
3. Traitement de la branche droite

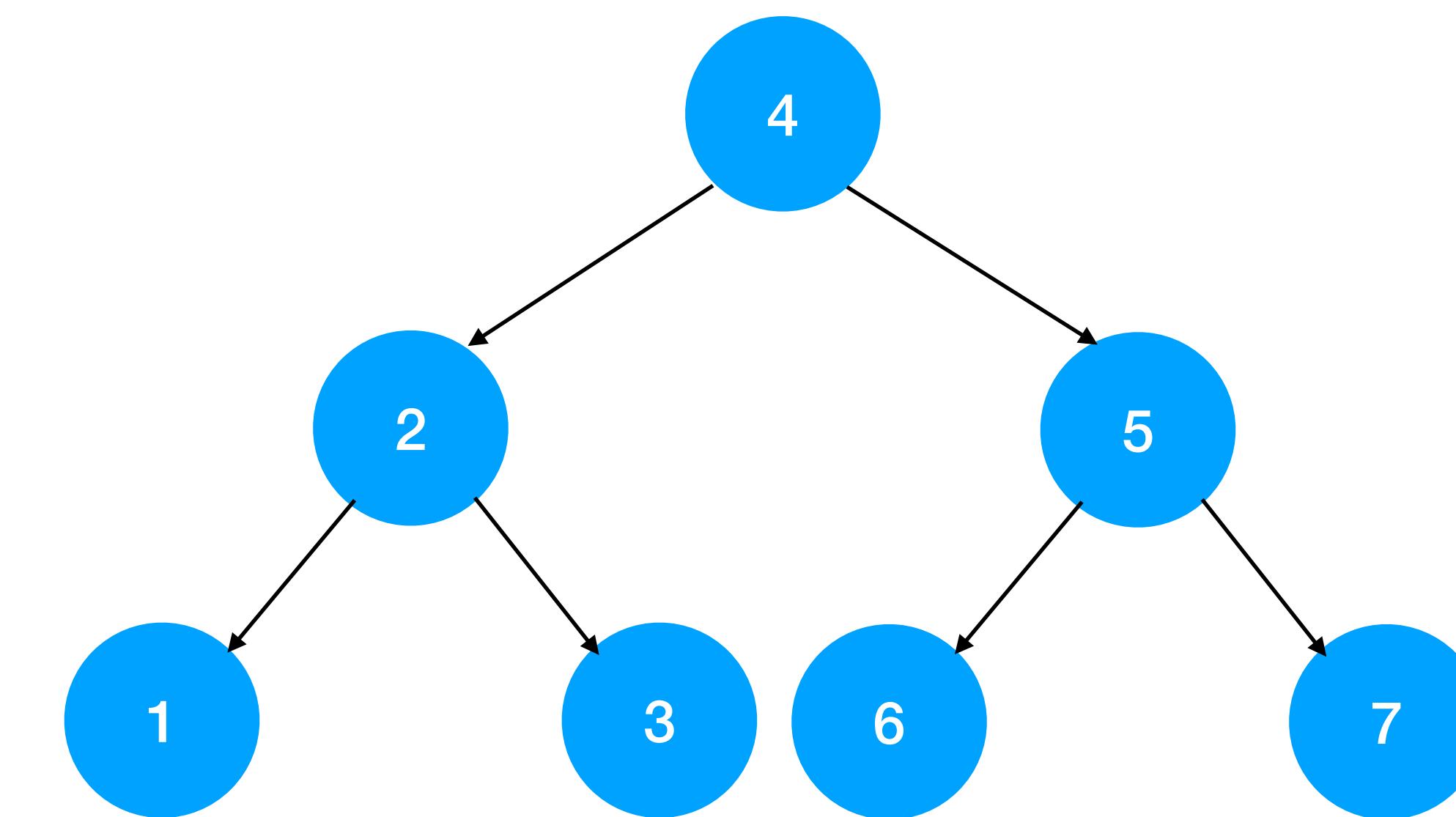
1, 2, 3, 4, 6, 5, 7



Question 3.1.10 : Parcours postfix

Parcours postfix :

- 1. Traitement de la branche gauche**
- 2. Traitement de la branche droite**
- 3. Traitement de la racine**

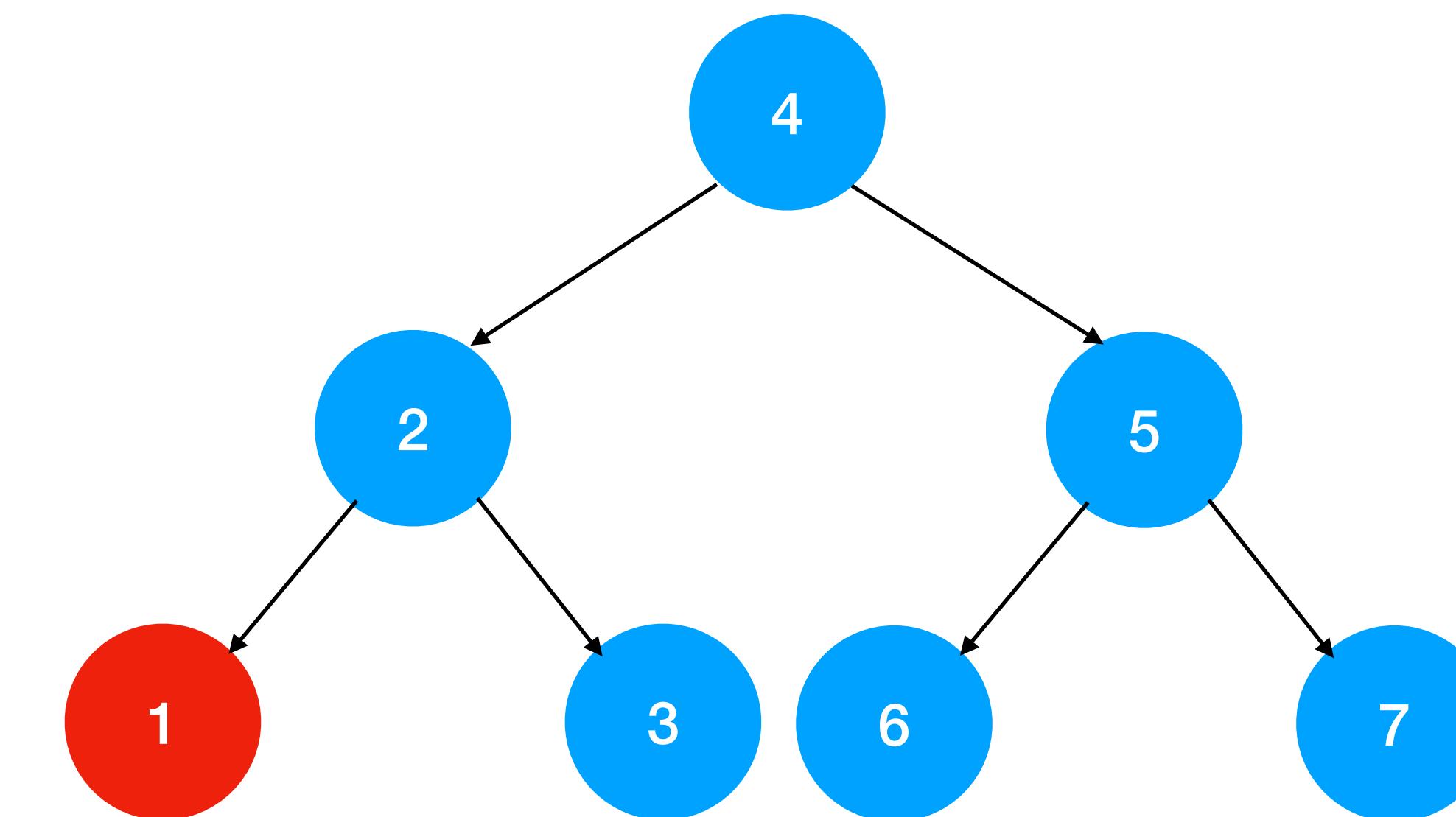


Question 3.1.10 : Parcours postfix

Parcours postfix :

- 1. Traitement de la branche gauche**
- 2. Traitement de la branche droite**
- 3. Traitement de la racine**

1

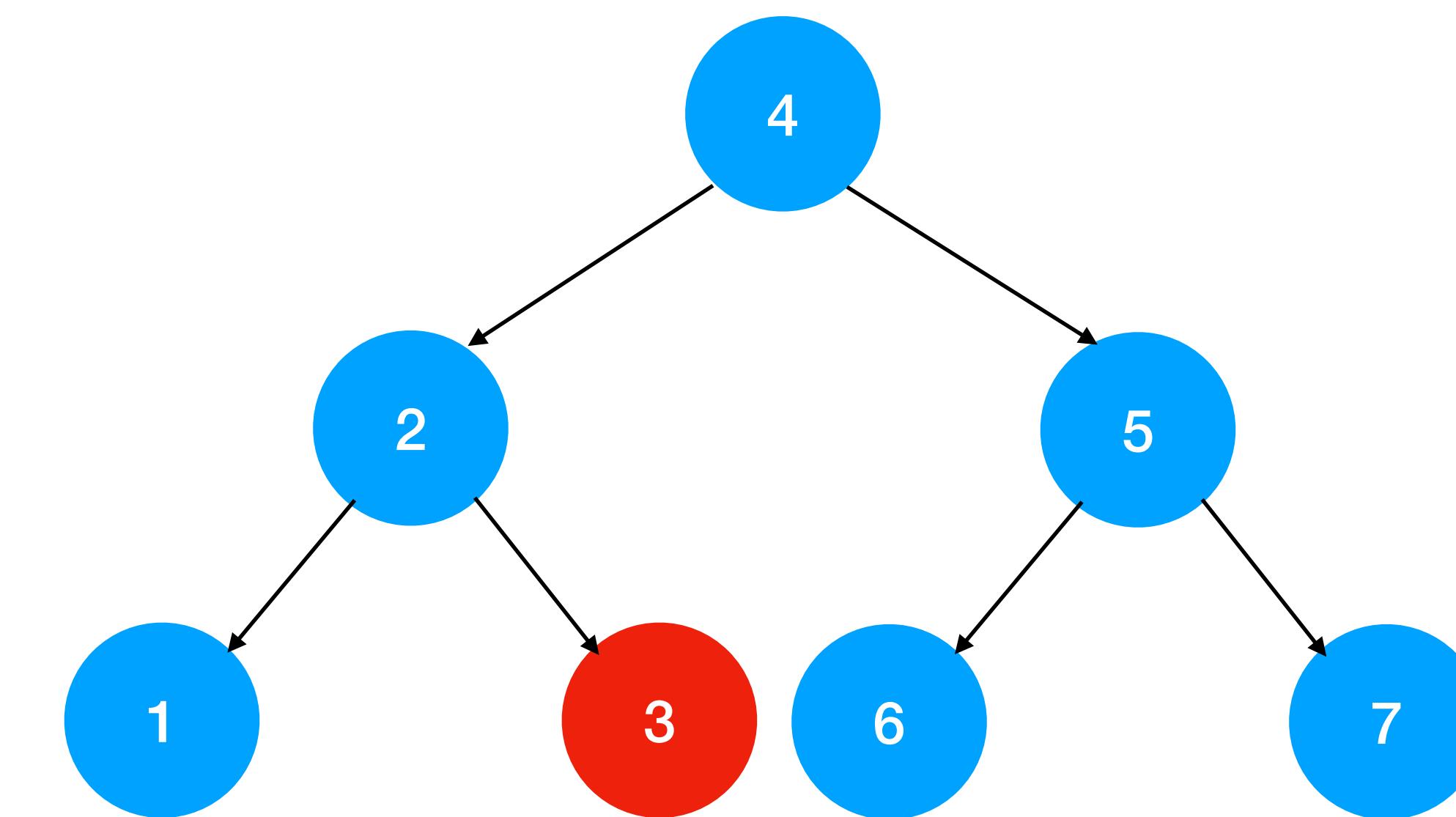


Question 3.1.10 : Parcours postfix

Parcours postfix :

- 1. Traitement de la branche gauche**
- 2. Traitement de la branche droite**
- 3. Traitement de la racine**

1, 3

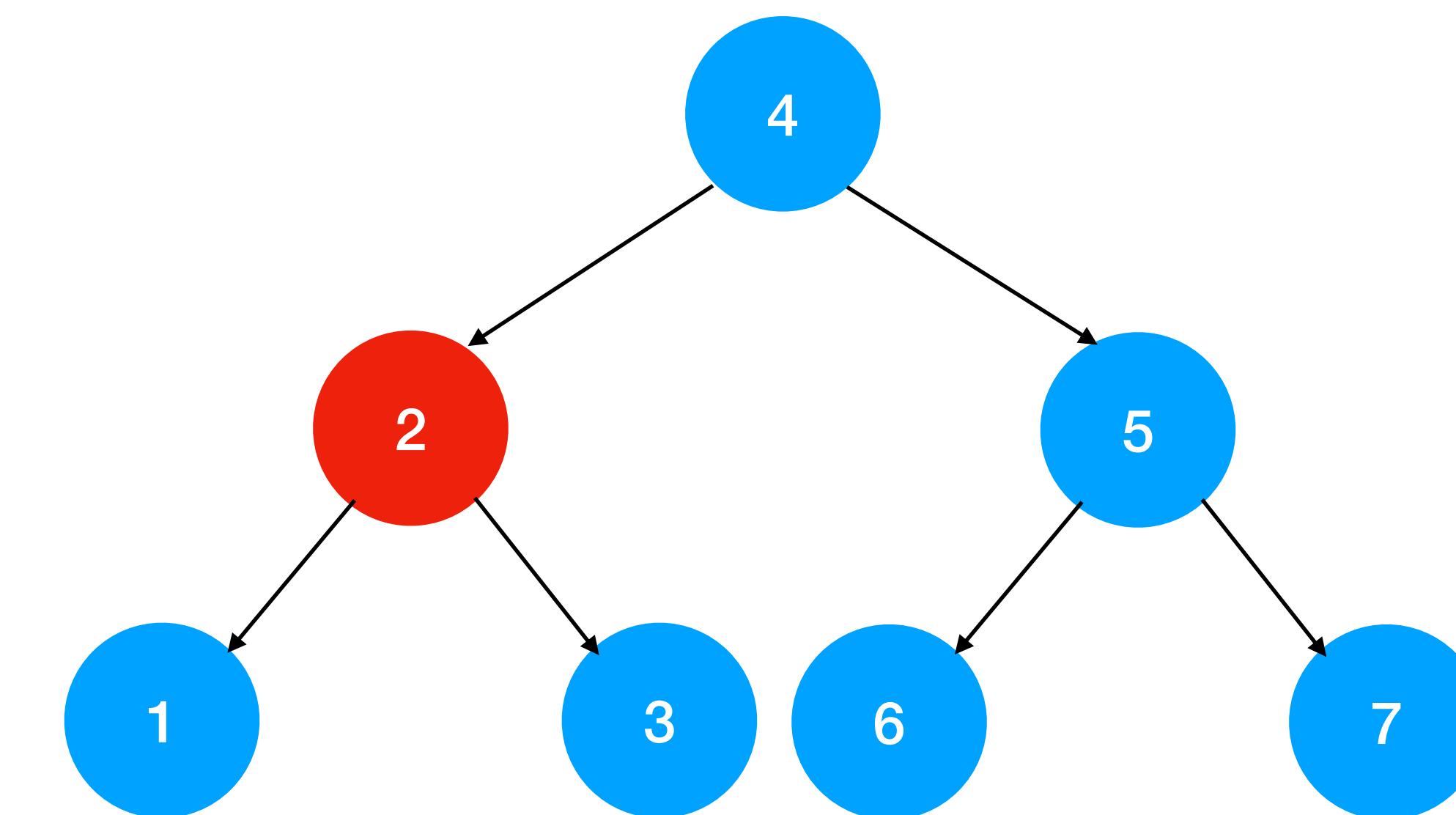


Question 3.1.10 : Parcours postfix

Parcours postfix :

- 1. Traitement de la branche gauche**
- 2. Traitement de la branche droite**
- 3. Traitement de la racine**

1, 3, 2

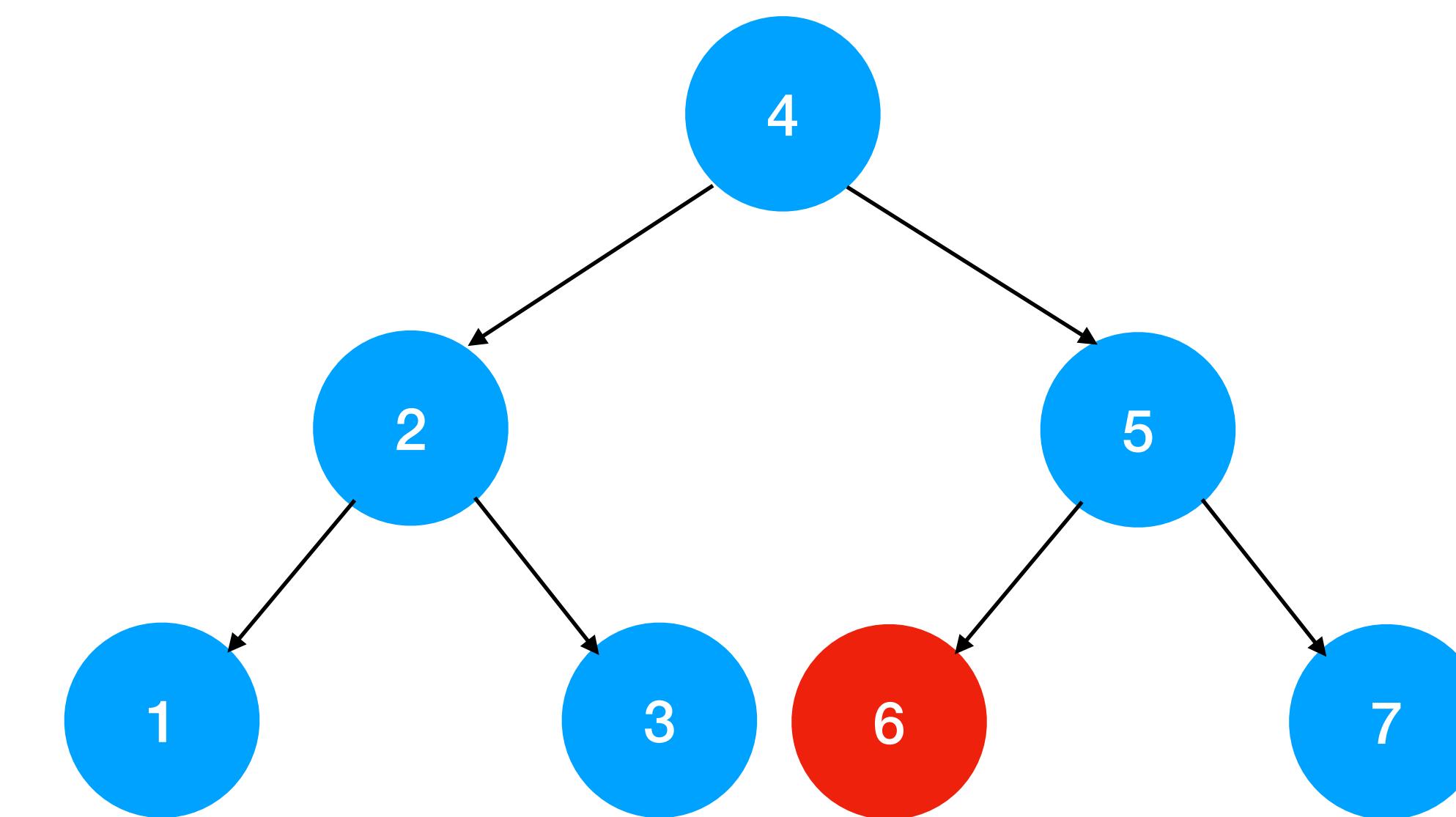


Question 3.1.10 : Parcours postfix

Parcours postfix :

- 1. Traitement de la branche gauche**
- 2. Traitement de la branche droite**
- 3. Traitement de la racine**

1, 3, 2, 6

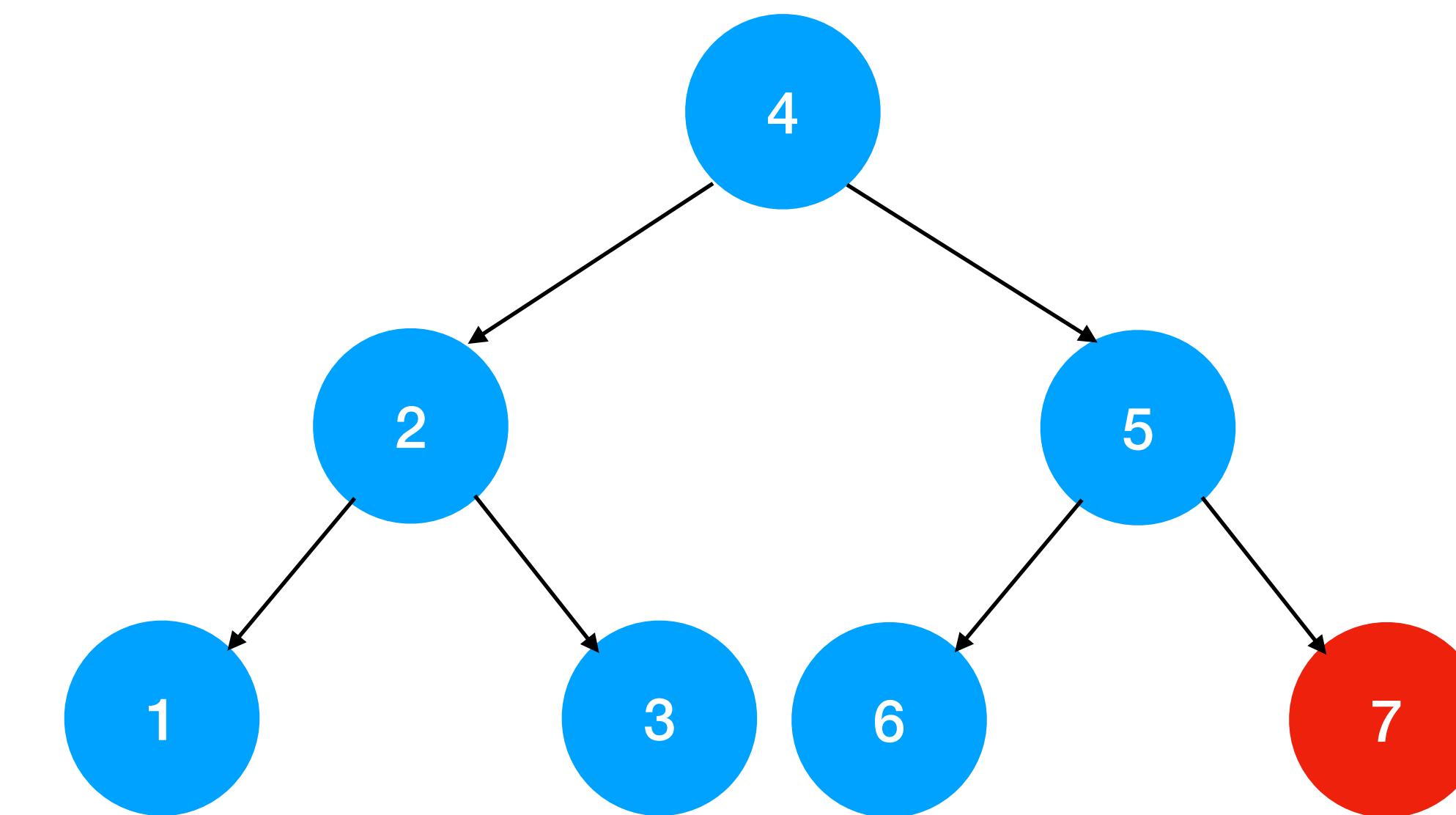


Question 3.1.10 : Parcours postfix

Parcours postfix :

- 1. Traitement de la branche gauche**
- 2. Traitement de la branche droite**
- 3. Traitement de la racine**

1, 3, 2, 6, 7



Question 3.1.10 : Parcours postfix

Parcours postfix :

- 1. Traitement de la branche gauche**
- 2. Traitement de la branche droite**
- 3. Traitement de la racine**

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

