Sur les arbres ABR

Objectif :

Découvrir chez les **arbres** certaines propriétés intéressantes :

- la **recherche** dans une liste peut être traitée avec des arbres.
- les arbres ont un intérêt lorsque la liste **évolue rapidement.**
- les arbres permettent de gérer de **gros volumes** de données.
- les ABR réalisent de manière **efficace** les opérations de base: recherche, insertion, suppression, fusion.

Qu'est-ce qu'un ABR ?

Un arbre binaire **B** est un **Arbre Binaire de Recherche** (ABR) si:

- soit **B** est vide,
- soit **B** est non vide et, si **v** est la valeur associée à sa racine, on a:
  - toute valeur **x** associée à son sous arbre **gauche** est telle que:

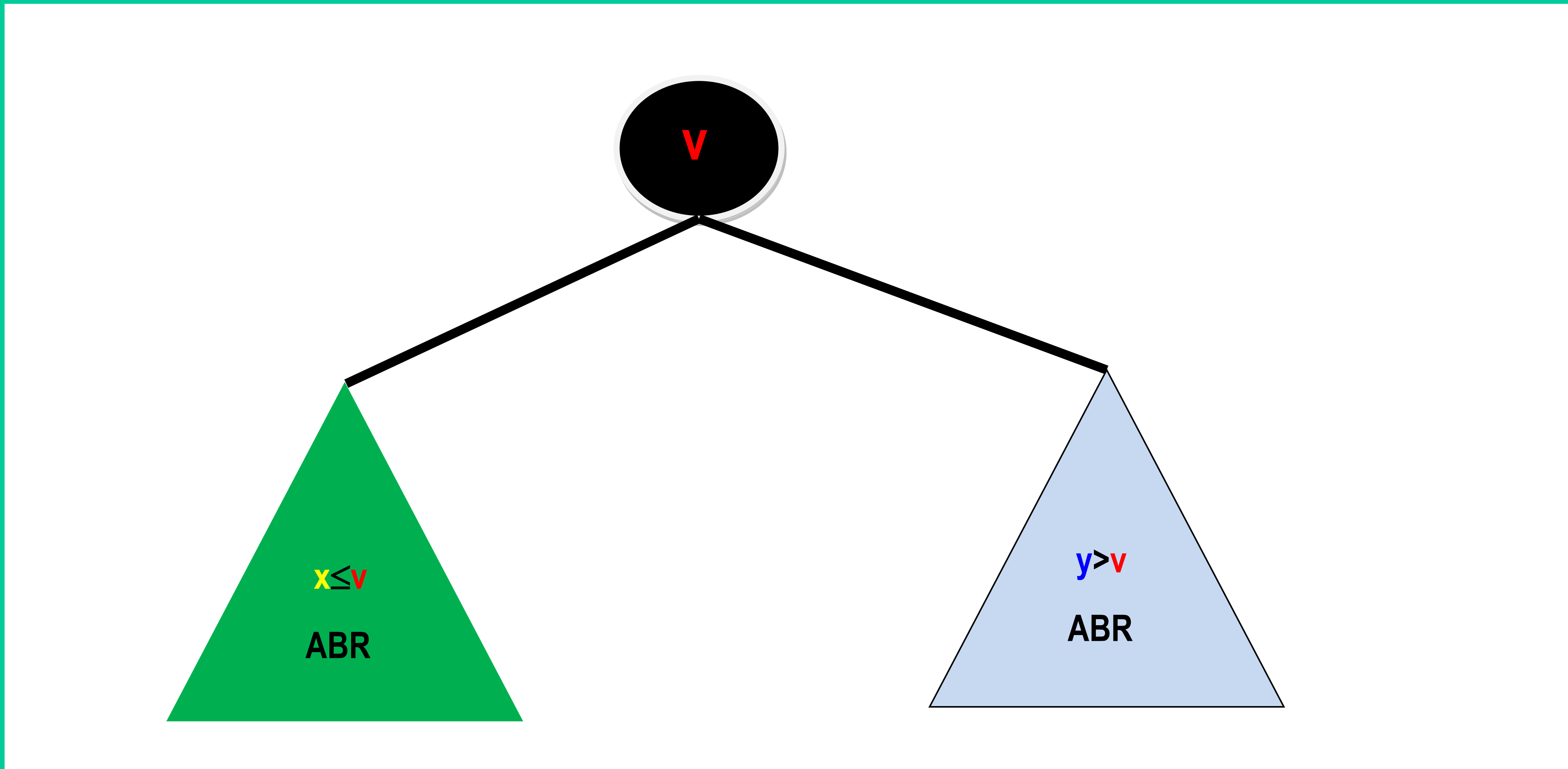
    $$x \leq v$$

  - toute valeur y associée à son sous arbre **droit** est telle que:

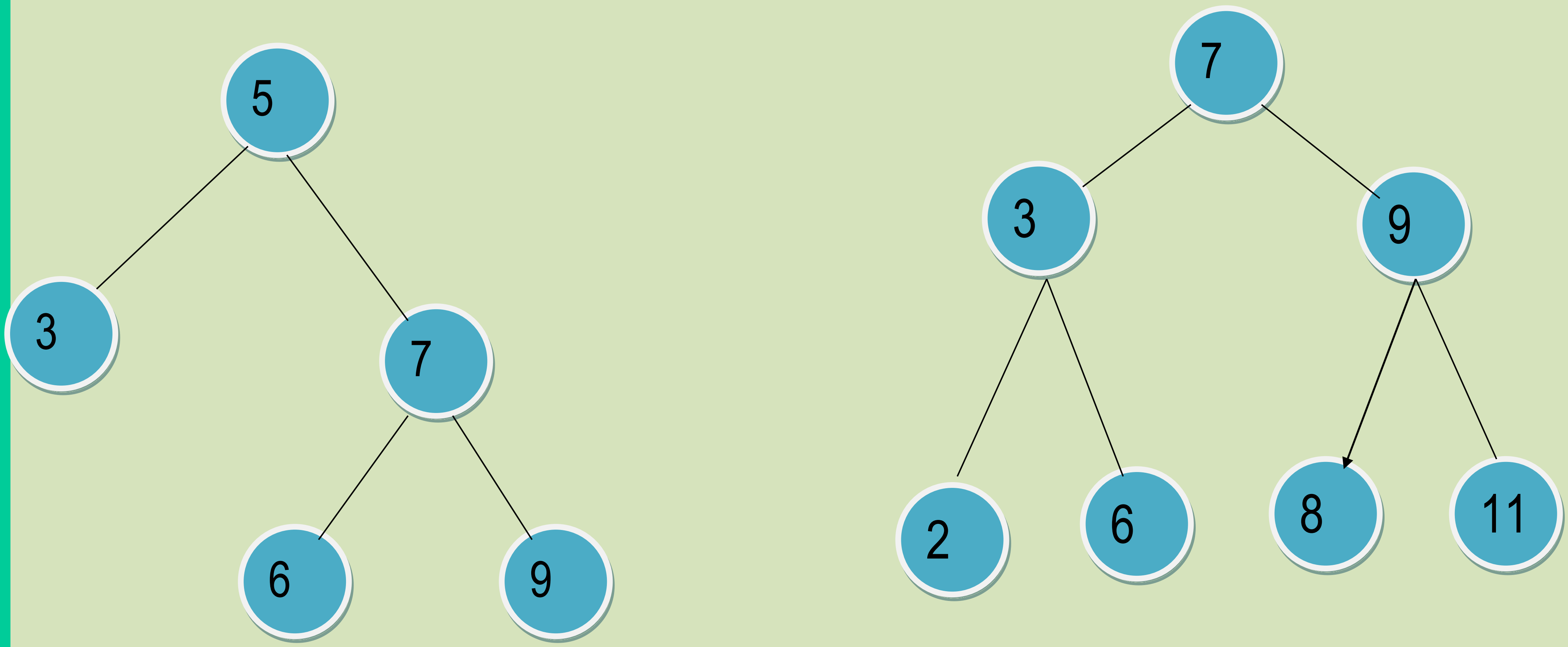
    $$y > v$$

  - tout sous arbre de **B** est un ABR.

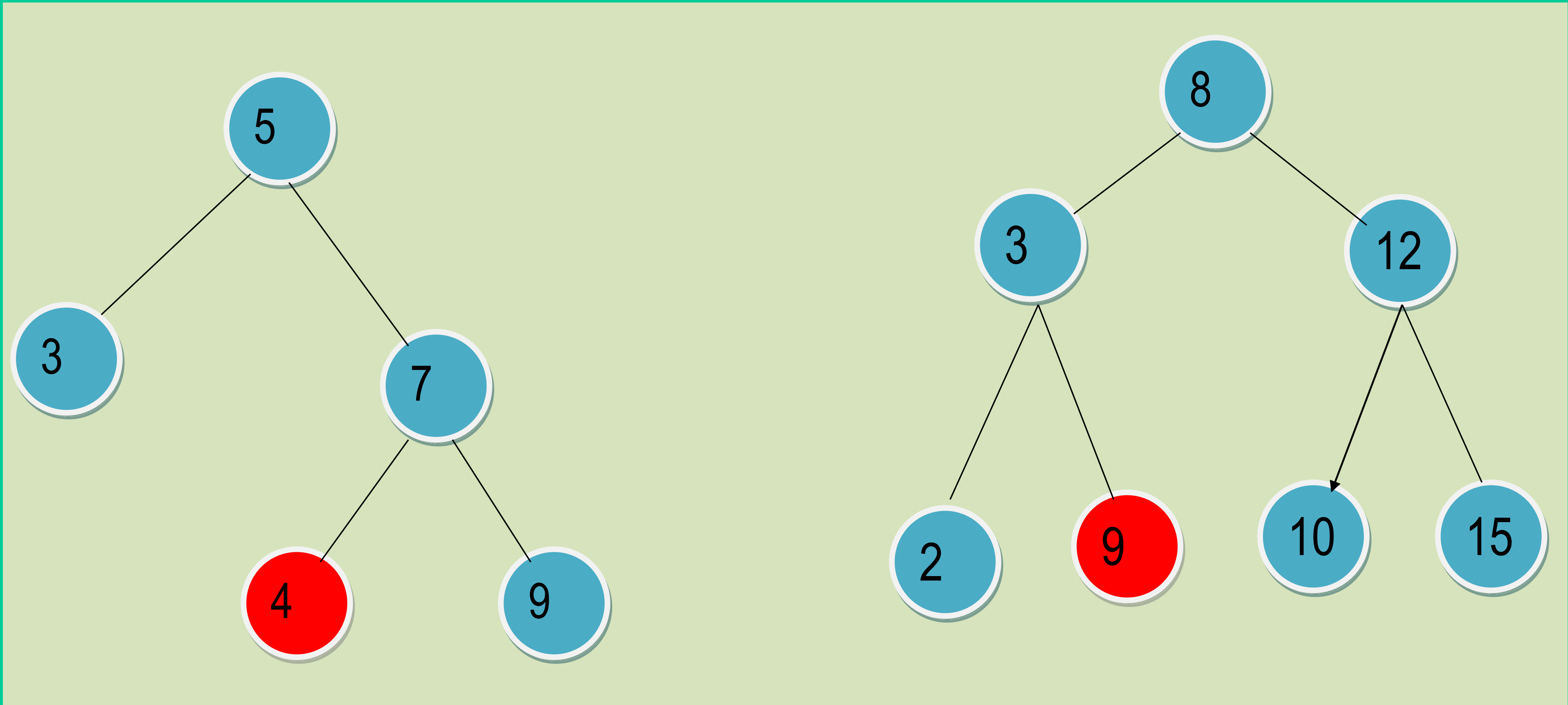
Illustration simple d'un ABR



Exemples d’arbres ABR



Exemples d’arbres non ABR



Remarques importantes

1-Les opérations de base sur les ABR ont un coût en temps proportionnel à la **hauteur** de l'arbre.

2-La **hauteur** d'un ABR de taille **n** construit aléatoirement est de l'ordre de $\log_2 n$

3-Pour les **H-équilibrés** ou arbres **AVL**(**A**delson-**V**elsky et **L**andis):

$$\textbf{hauteur} \leq \log_2 (n)$$

3-D’après ce qui précède, pour un ABR de taille n, les opérations de base requièrent en moyenne un temps en **$\log_2 n$**

4-Lorsque l’arbre **dégénère en liste** de longueur **n**, ces mêmes opérations requièrent un temps linéaire en **n** : c’est le **cas pire**.

Recherche dans un ABR

```
Trouvé?(x,B) :Booléen
begin
    if estArbreVide(B) then retourner faux
        else
        if  racine(B) = x then retourner  vrai
                        else
                        if racine(B) > x  then  Trouvé?(x, gauche(B))
                                             else  Trouvé?(x, droit(B))
end
```

Suppression d'un nœud dans un ABR

Trois cas se présentent selon le statut du nœud :

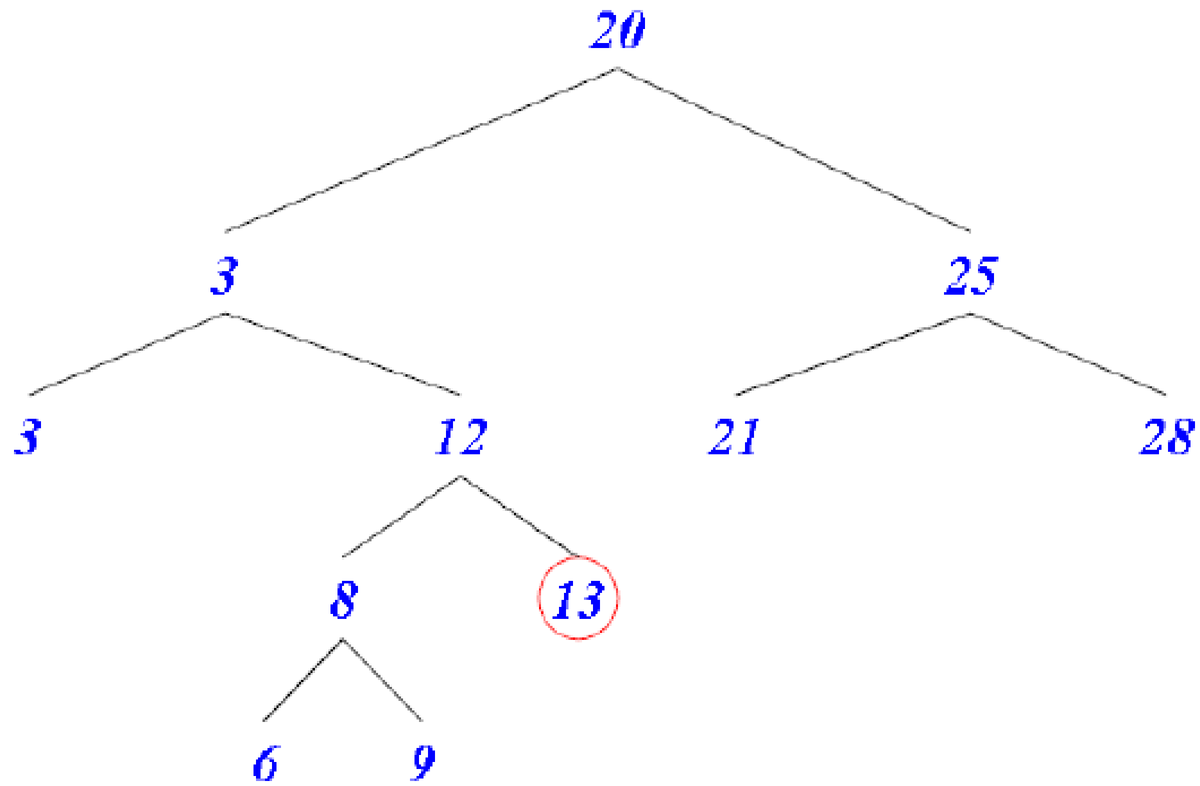
1- le nœud **X** est une **feuille**: supprimer **x**

2- le nœud **X** a **un fils unique** **F**: remplacer **x** par **F**

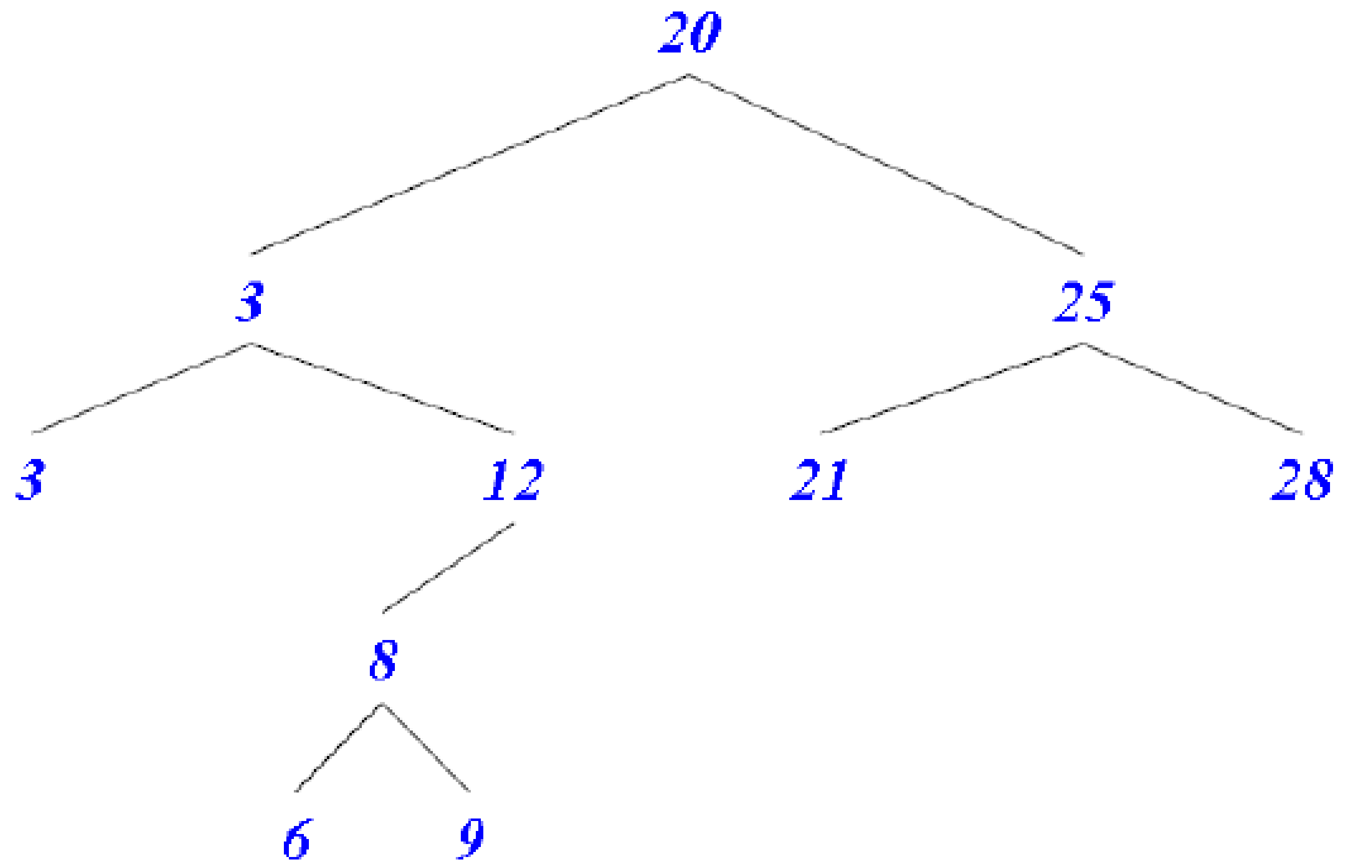
3- le nœud **X** **a deux fils**:

- chercher le nœud maximum **M**: le plus à droite dans le sous-arbre gauche de **X** ,
- remplacer **X** par **M**,
- supprimer **M** qui n'a pas de fils droit.

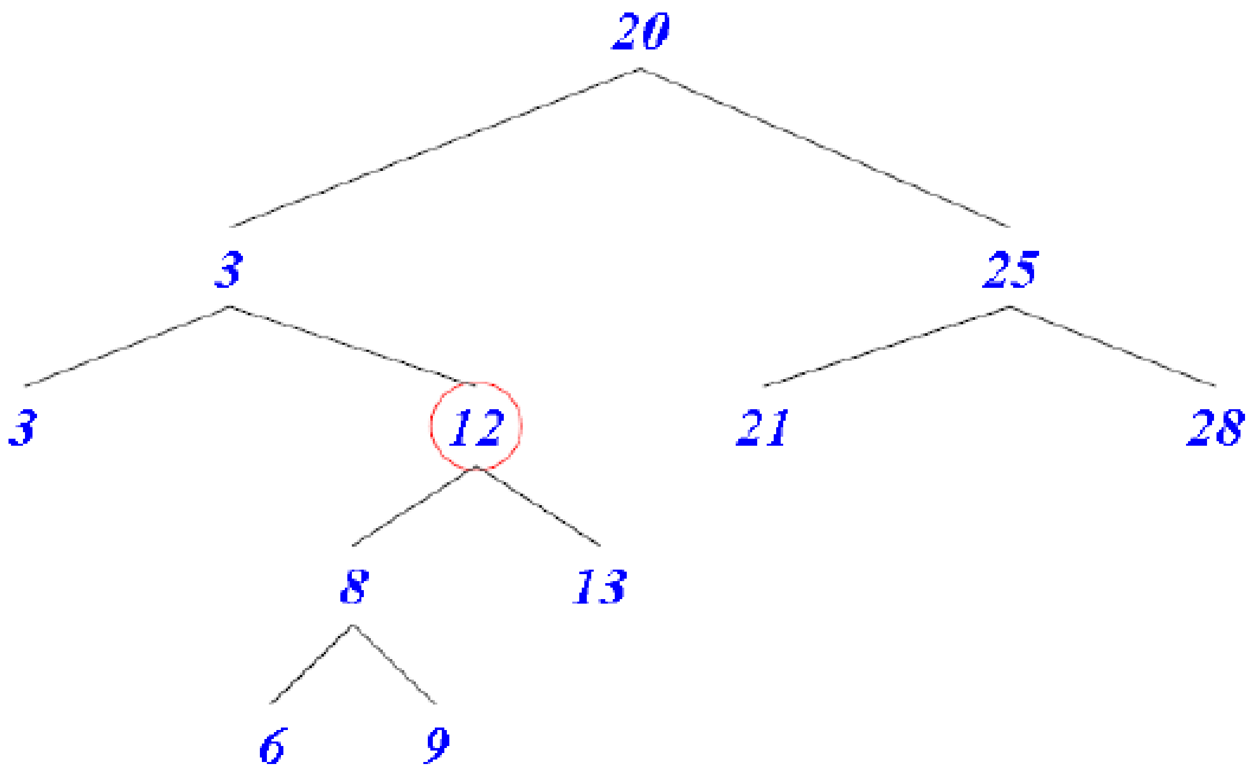
Exemple 1 :supprimer la feuille 13



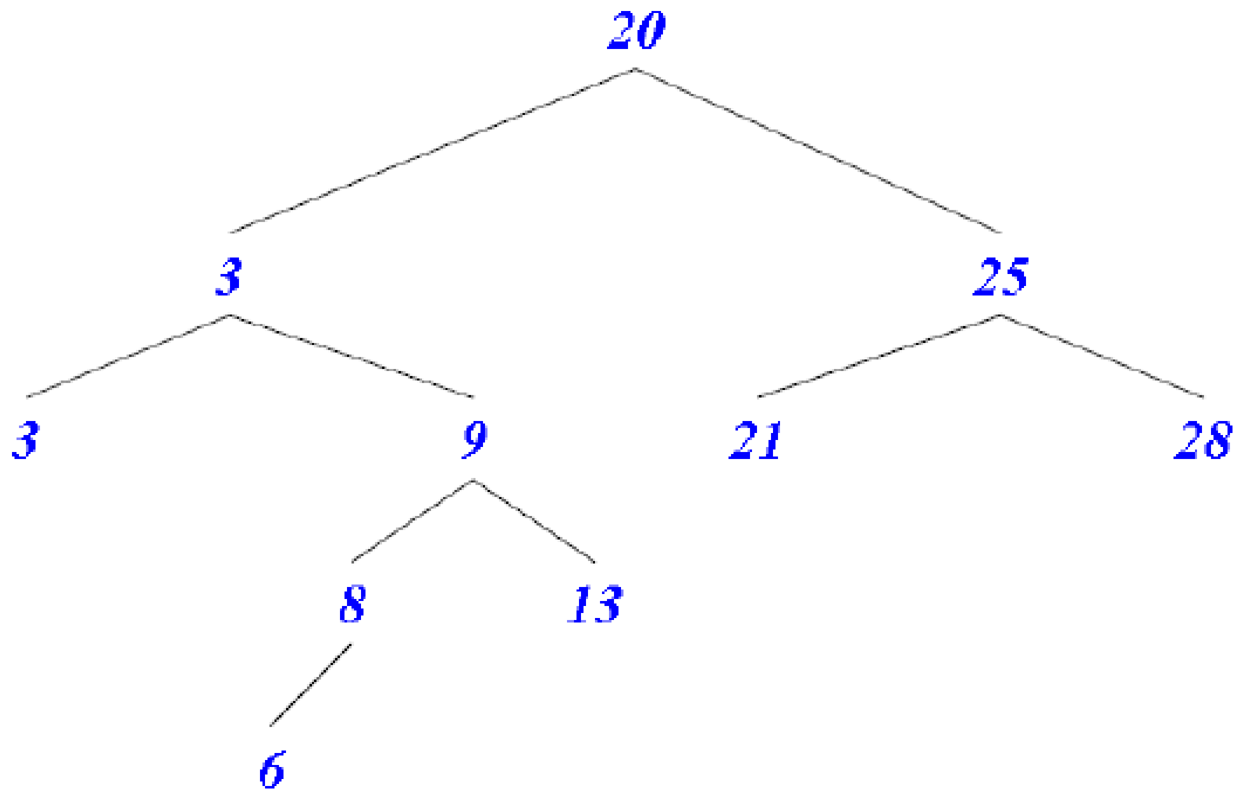
On supprime le 13


20
3
25
3
12
21
28
8
6
9

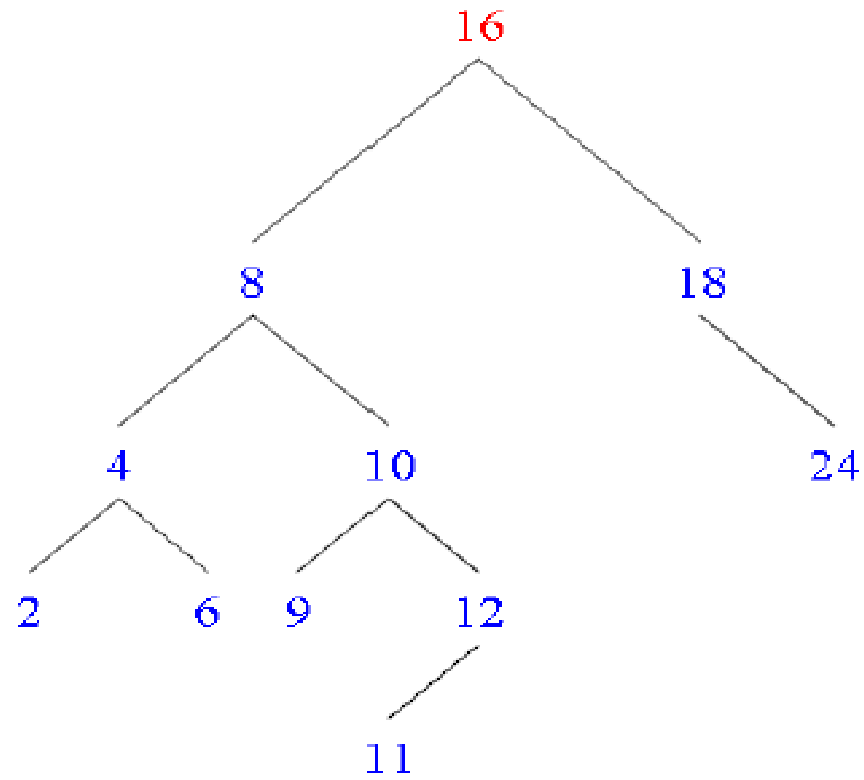
Exemple 2 : supprimer le nœud intermédiaire 12



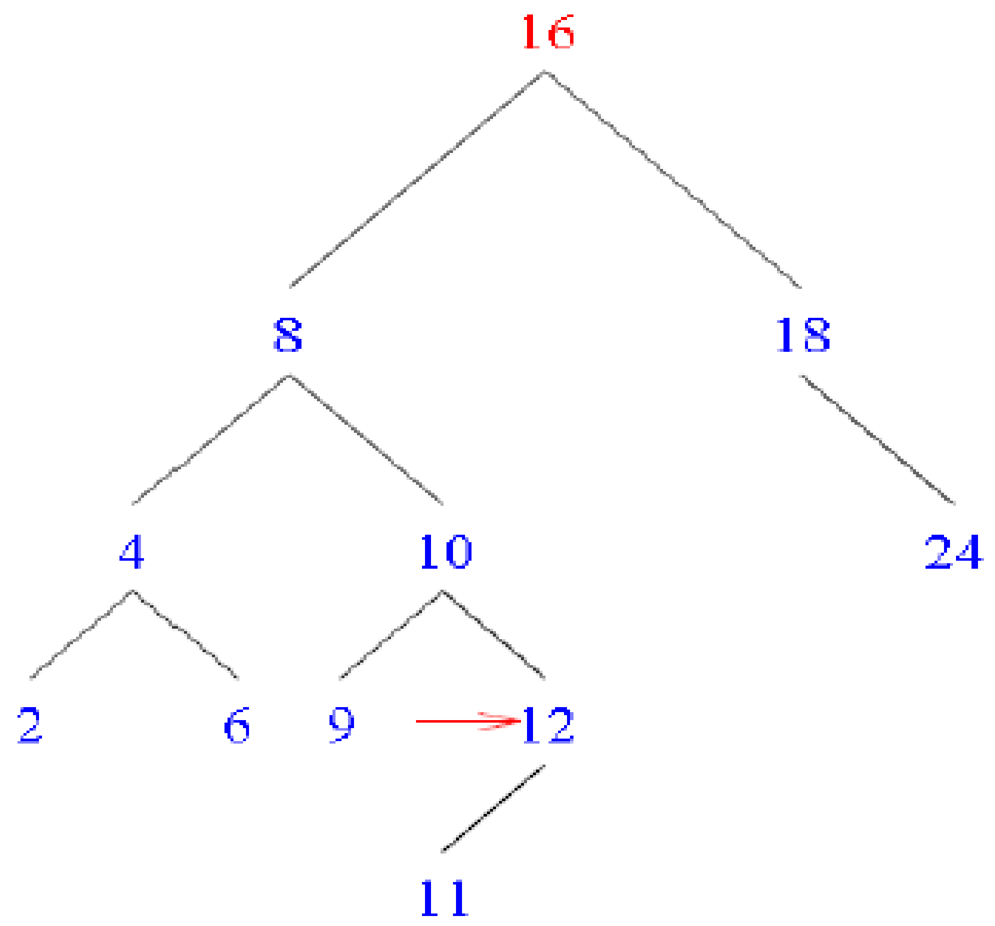
On supprime le 12

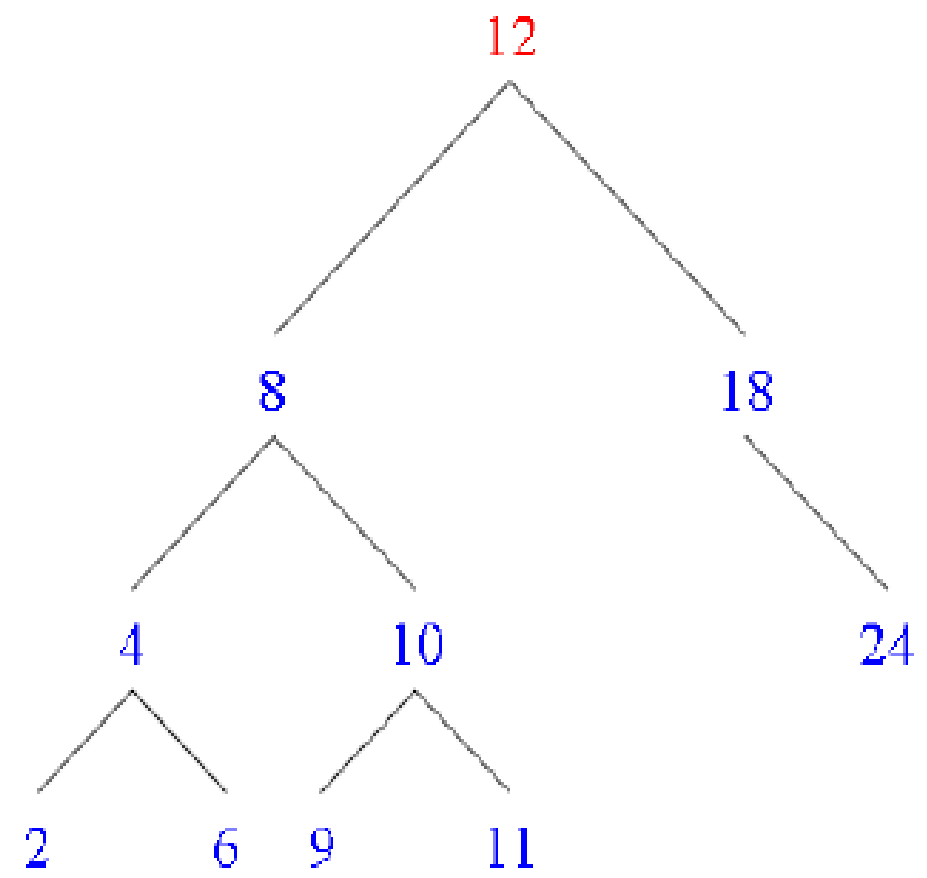

20
3
25
3
9
21
28
8
13
6

Exemple 3 : supprimer la racine 16



On supprime le *16*


16
8
18
4
10
24
2
6
9
12
11


12
8
18
4
10
24
2
6
9
11

Supprimer le nœud maximum d'un ABR

```
supMax(A, Max)
Debut
    if  droit(A) = arbreVide
            then        Max := racine(A);
                        A := gauche(A)
            else        supMax(droit(A), Max)
end
```

Supprimer le nœud minimum d'un ABR

```
supMin(A, Min)
Debut
    if  gauche(A) = ArbreVide
            then        Min := racine(A)
                         A := droit(A)
            else supMin(gauche(A), Min)
end
```

Supprimer un nœud X

```
supX(A,X)
begin
    if A ≠ arbreVide then
/* cas de suppression dans arbre gauche */
                if  X < racine(A)          then supX(gauche(A),X)

/* cas de suppression dans arbre droit*/
             else  if X > racine(A)   then supX(droit(A),X)
```

```
/* cas de suppression à la racine : X= racine(A)    */
   else
/* cas gauche(A) vide   */
                 if gauche(A) = arbreVide
                                    then A := droit(A)

/* cas droite(A) vide   */
                 else if droit (A) =arbreVide
                                    then A := gauche(A)
```

```
                  else
 /* cas où ni gauche(A) ni droit(A) n'est vide :suppression du max à
gauche  */
                      begin
                      supMax(gauche(A),Max)  ;
                      racine(A) := Max
                      end
end
```

Ajouter un nœud à un ABR

Deux cas se présentent :

1-ajouter un **nœud feuille** :

.peut augmenter la **<u>hauteur</u>** de l'arbre

2-ajouter un **nœud racine** (cas général)

. modifie la **<u>structure</u>** de l'arbre

Rappel :

1- Calcul de la hauteur **H** d'un arbre:

$$H(\text{arbreVide}) = -1$$

$$H(A) = 1 + \text{Max}\ [\ H(\text{gauche}(A)),\ H(\text{droit}(A))]$$

Par abus de langage:

$$H(A) = H(\textbf{racine}(A))$$

2- Bornes **optimales**

Soit un arbre binaire A, non vide, de hauteur H et de taille n, on a :

$$[\, \log_2 \mathbf{n} \,] \leq H(A) \leq \mathbf{n-1}.$$

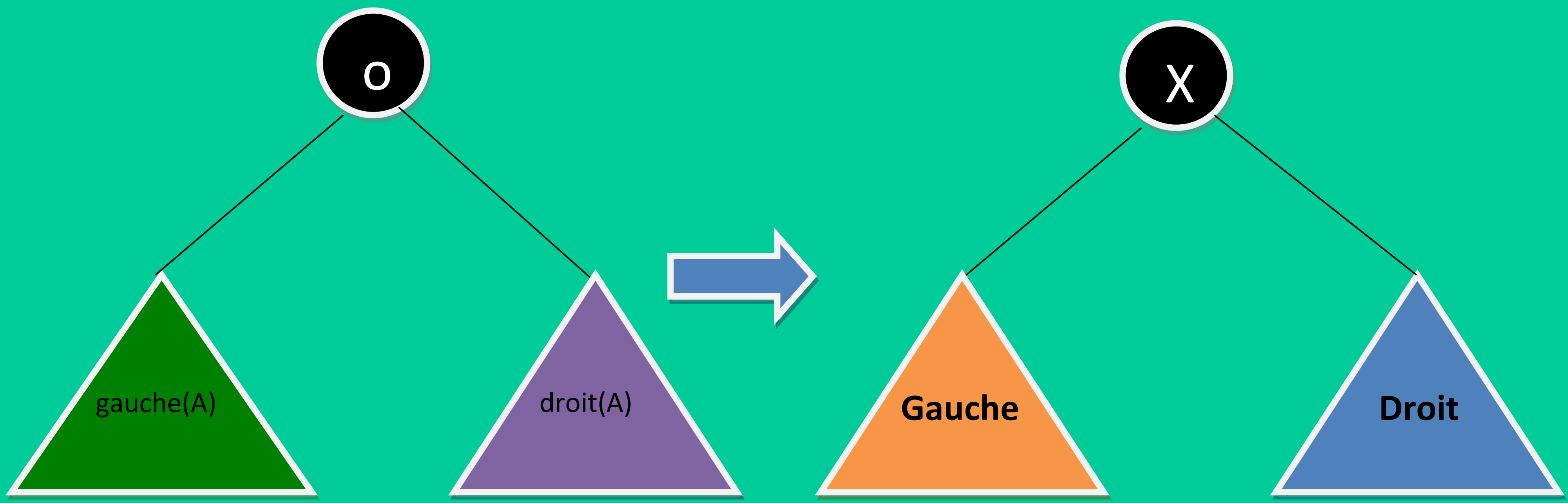
Insertion aux feuilles

```
ajouterFeuille(A,X)
begin
 if  A = arbreVide
        then   construire(X, foretVide)
        else  if X < racine(A) then AjouterFeuille(gauche(A),X)
                                    else AjouterFeuille(droit(A),X)
end
```

Insertion à la racine

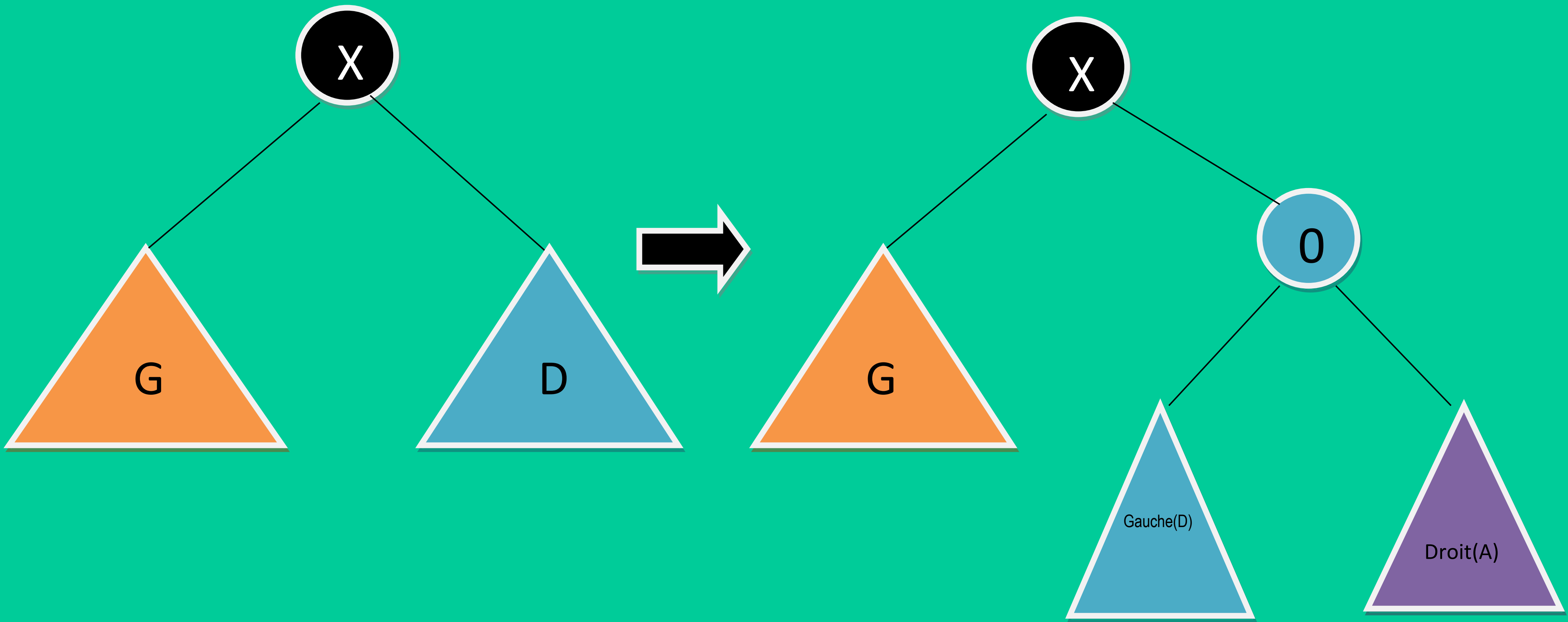
```
ajouterRacine(A,X)
 begin
        couper(X, A, Gauche, Droit ) ;
        laForet:= planter(Gauche,1, foretVide) ;
        laForet:= planter(Droit, 2, laForet)  ;
        A := construire(X, laForet)
end
```

Coupure de A suivant X

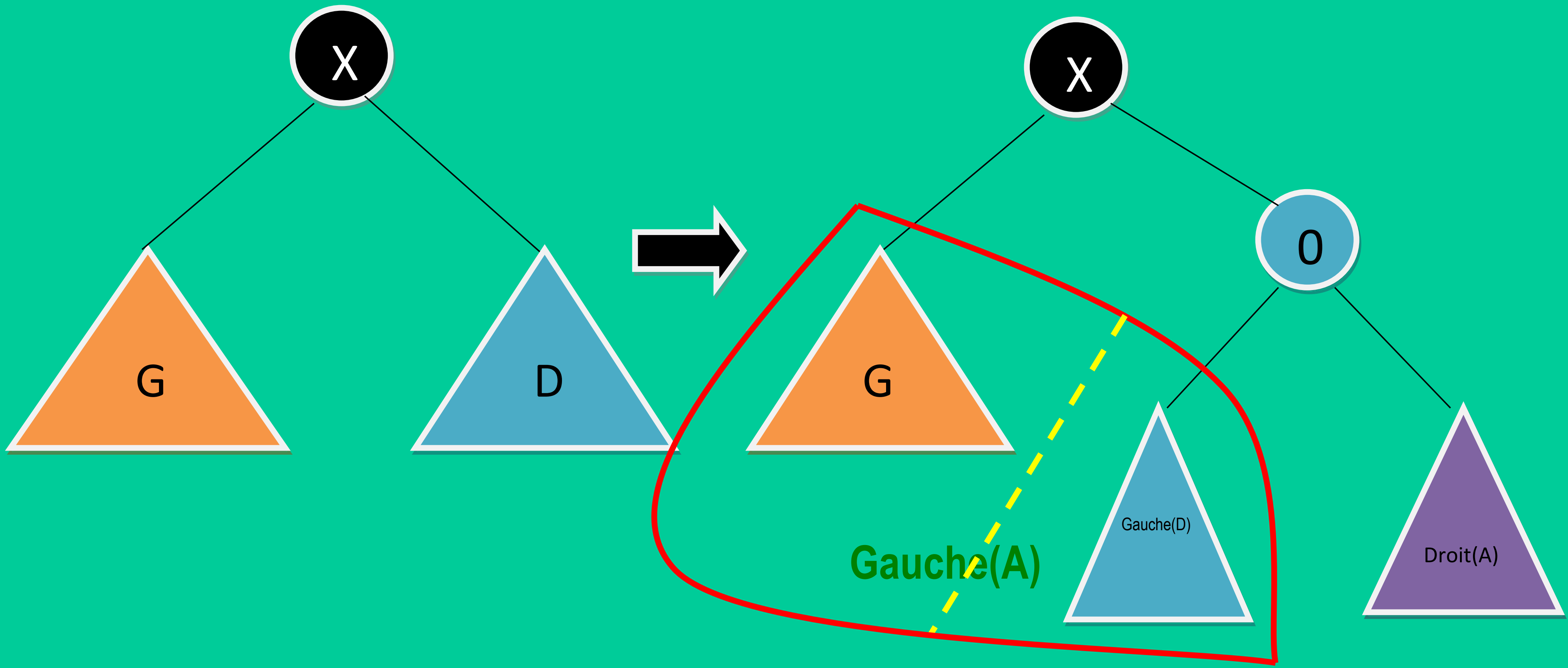


```
couper ( X, A, G, D)
begin
   if A = arbreVide then
/*cas où A est un arbre vide*/
                    G := arbreVide;
                    D:= arbreVide;
```

Cas X < racine(A)

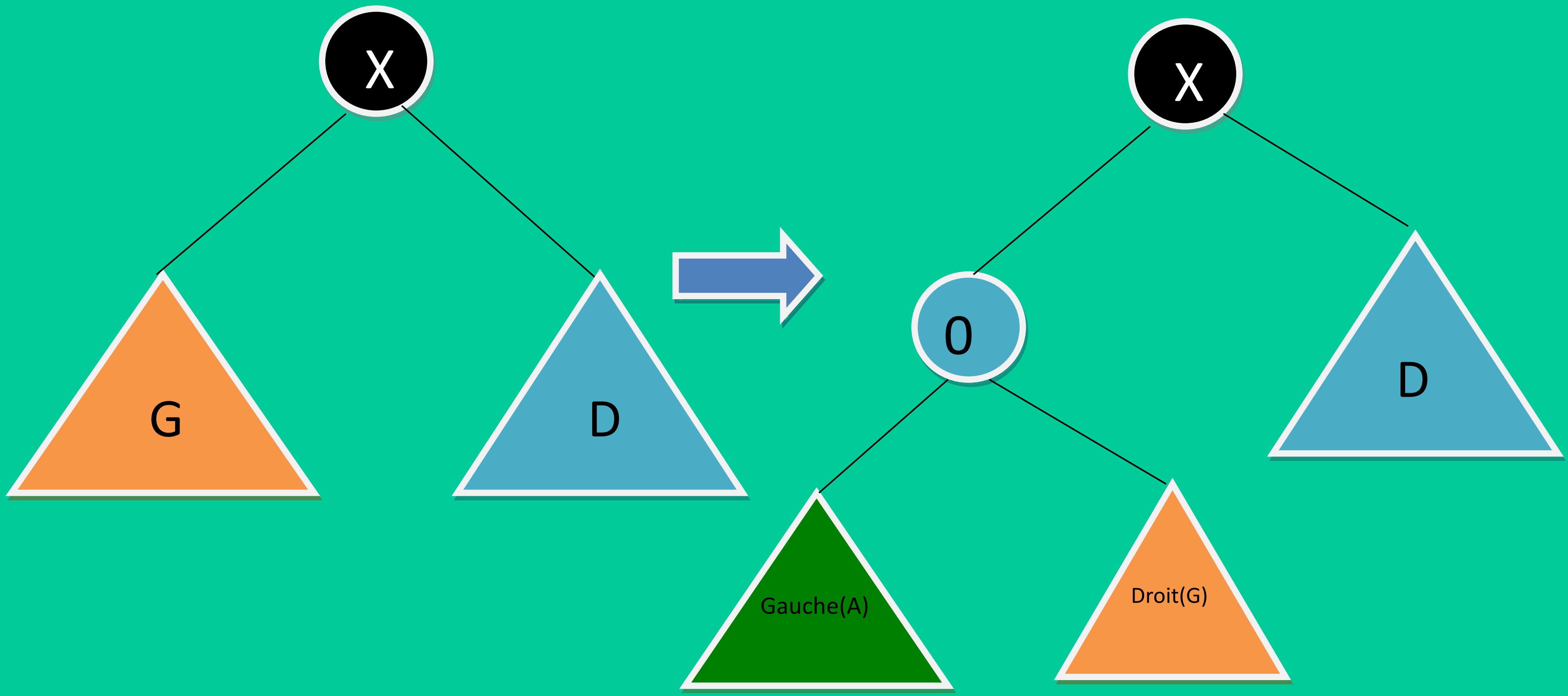


Cas X < racine(A)

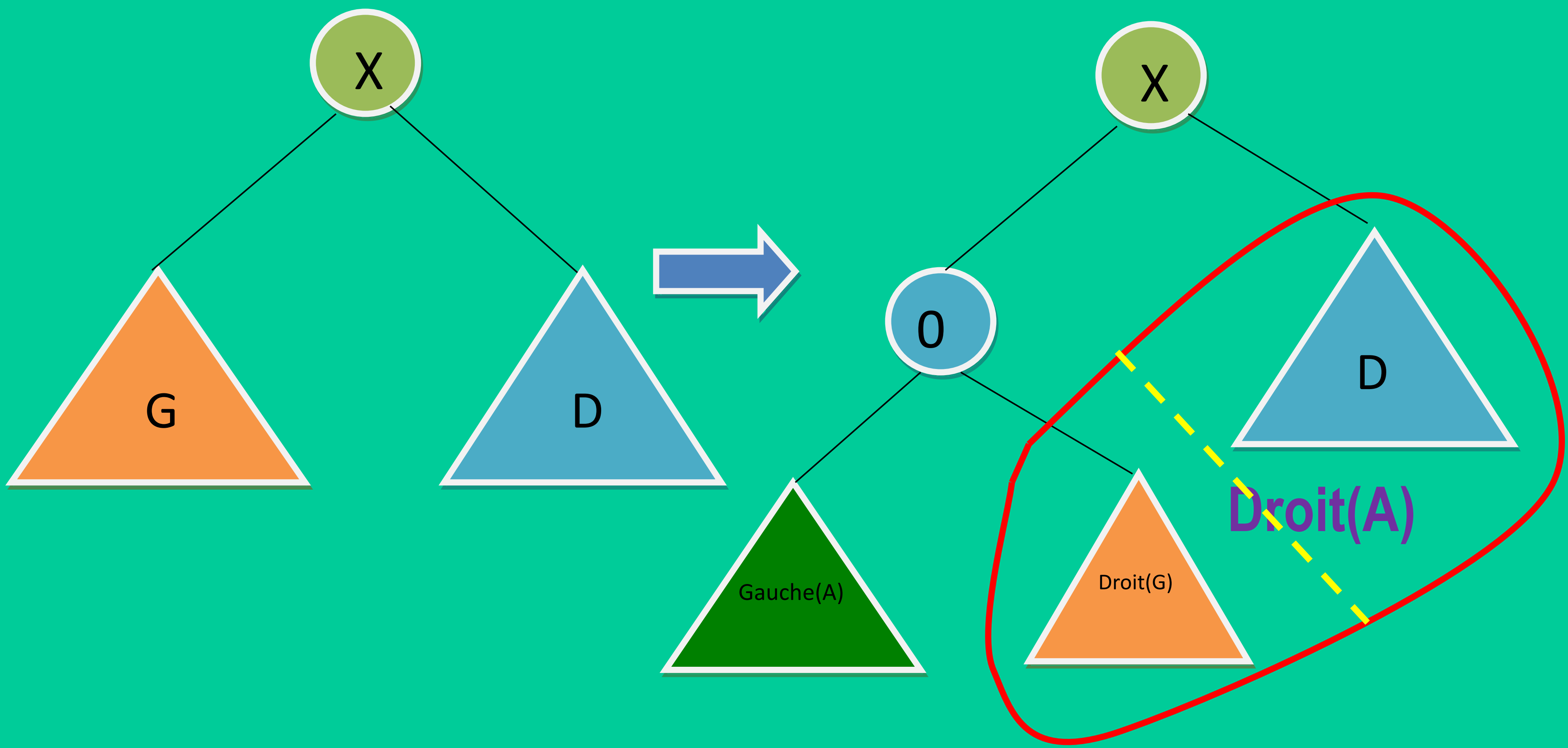


```
   else if  X < racine(A)
         then
/*couper gauche(A)*/
            droit(D):= droit(A)
            racine(D):= racine(A)
/* gauche(D)= sous-arbre de gauche(A) formé des nœuds  > X

            couper (X, gauche(A), G, gauche(D));
```

Cas X > racine(A)

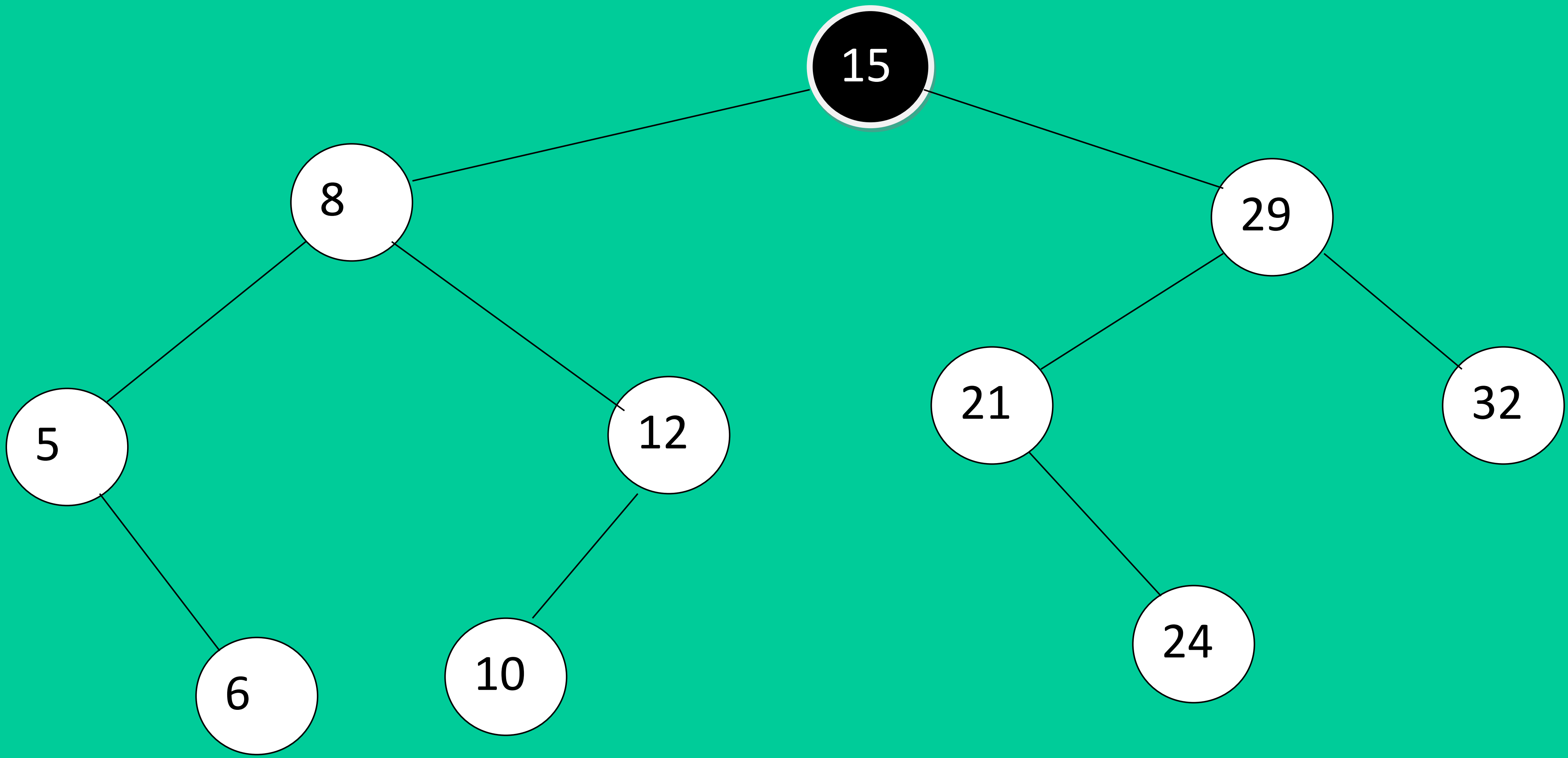


Cas X > racine(A)

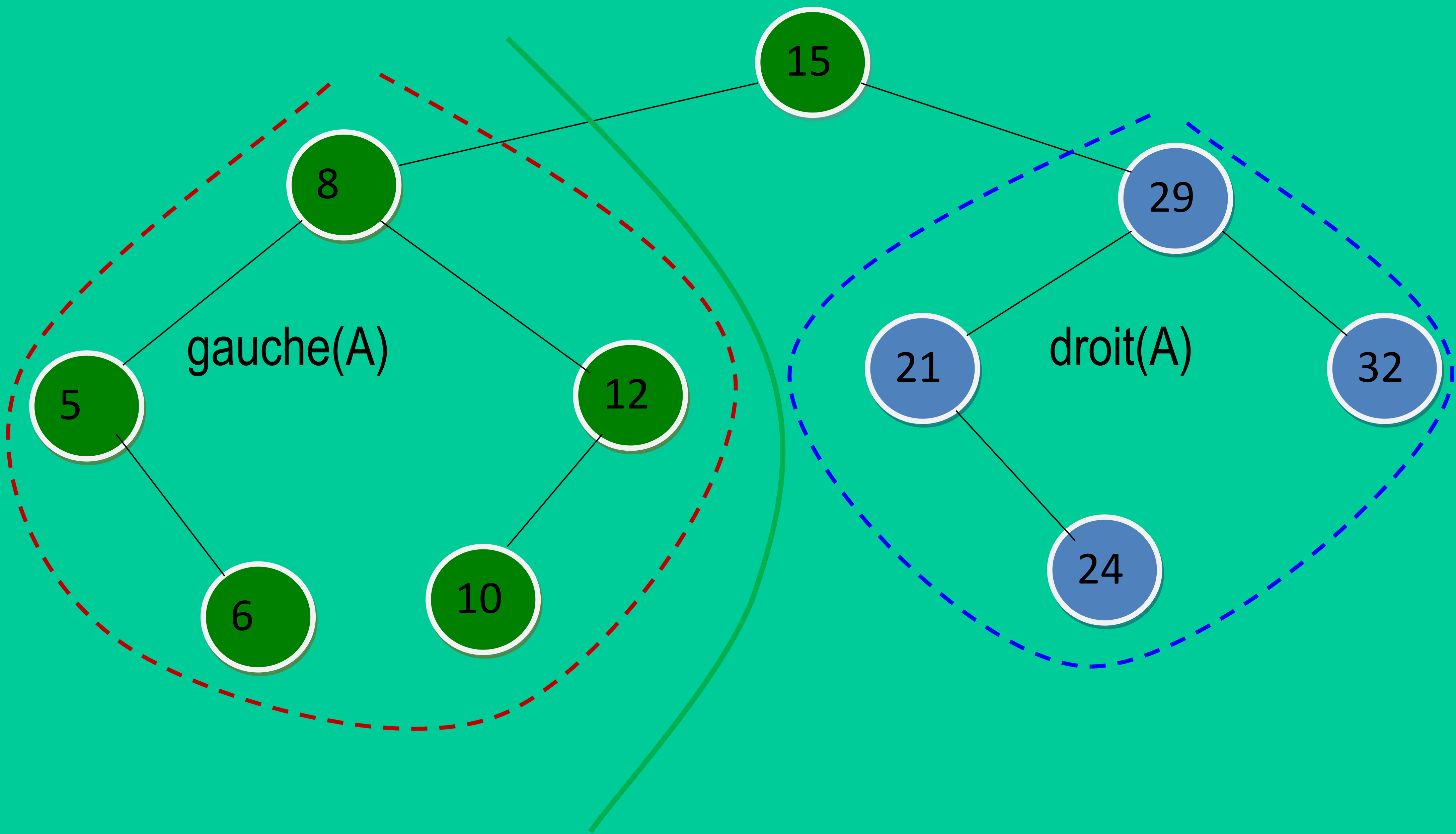


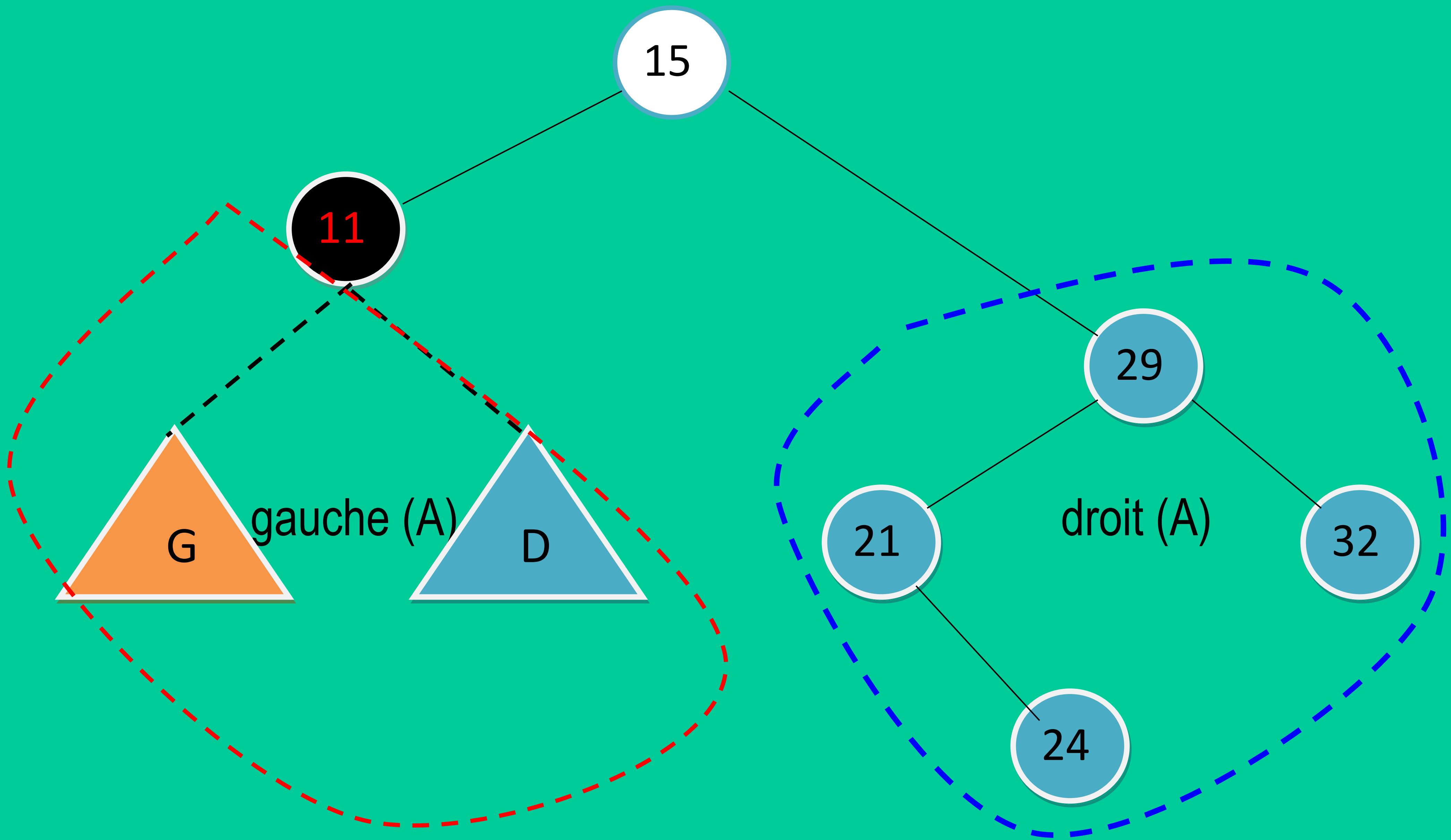
```
        else   /*couper le sous-arbre droit de A: droit(A)*/
            gauche (G):= gauche (A) ;
            racine (G):= racine (A) ;
/* droit(G) = sous-arbre de droit(A) formé des nœuds  < X

            couper (X, droit(A), droit(G), D);
end
```

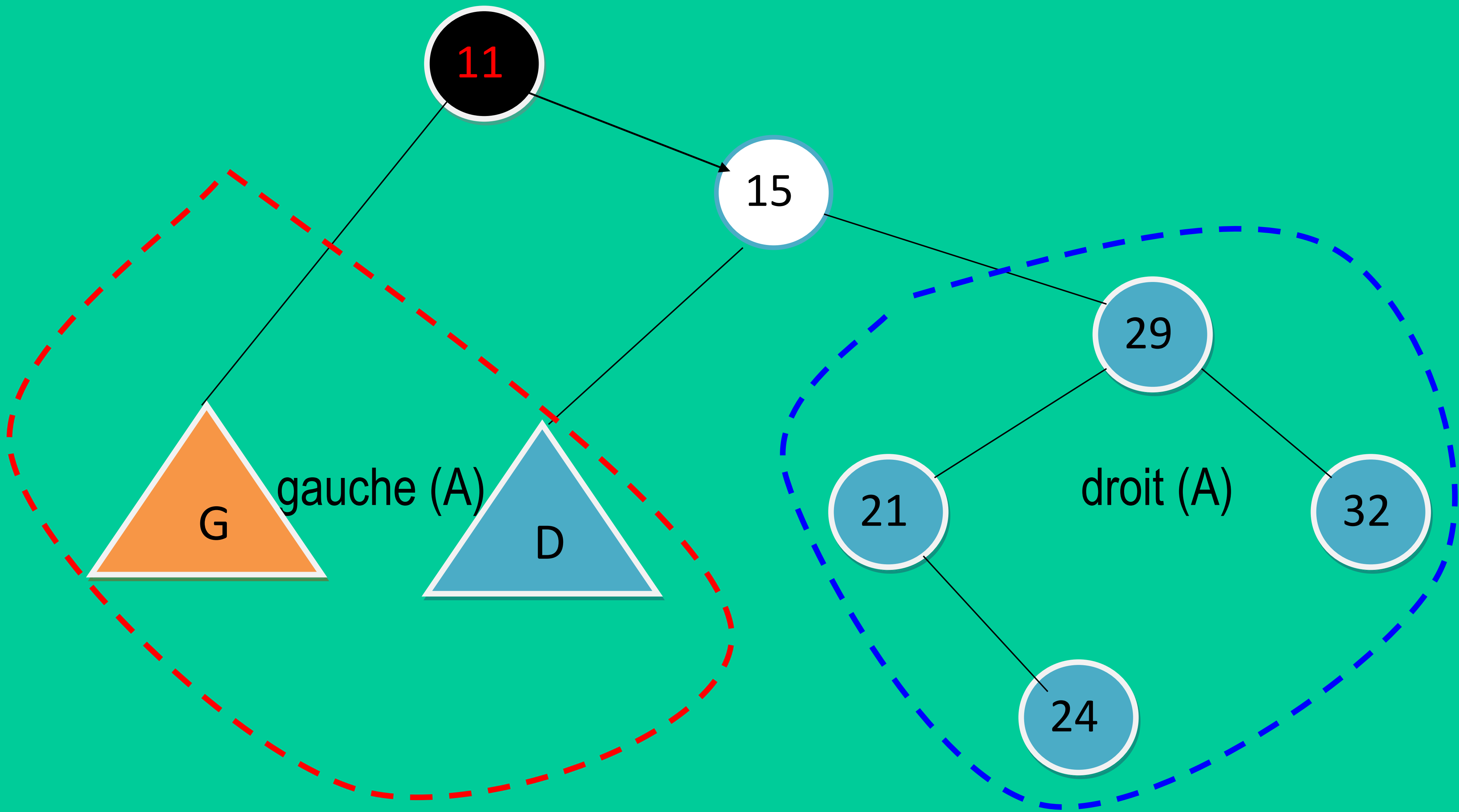
Exemple: Ajouter 11 à l'ABR ci-dessous :



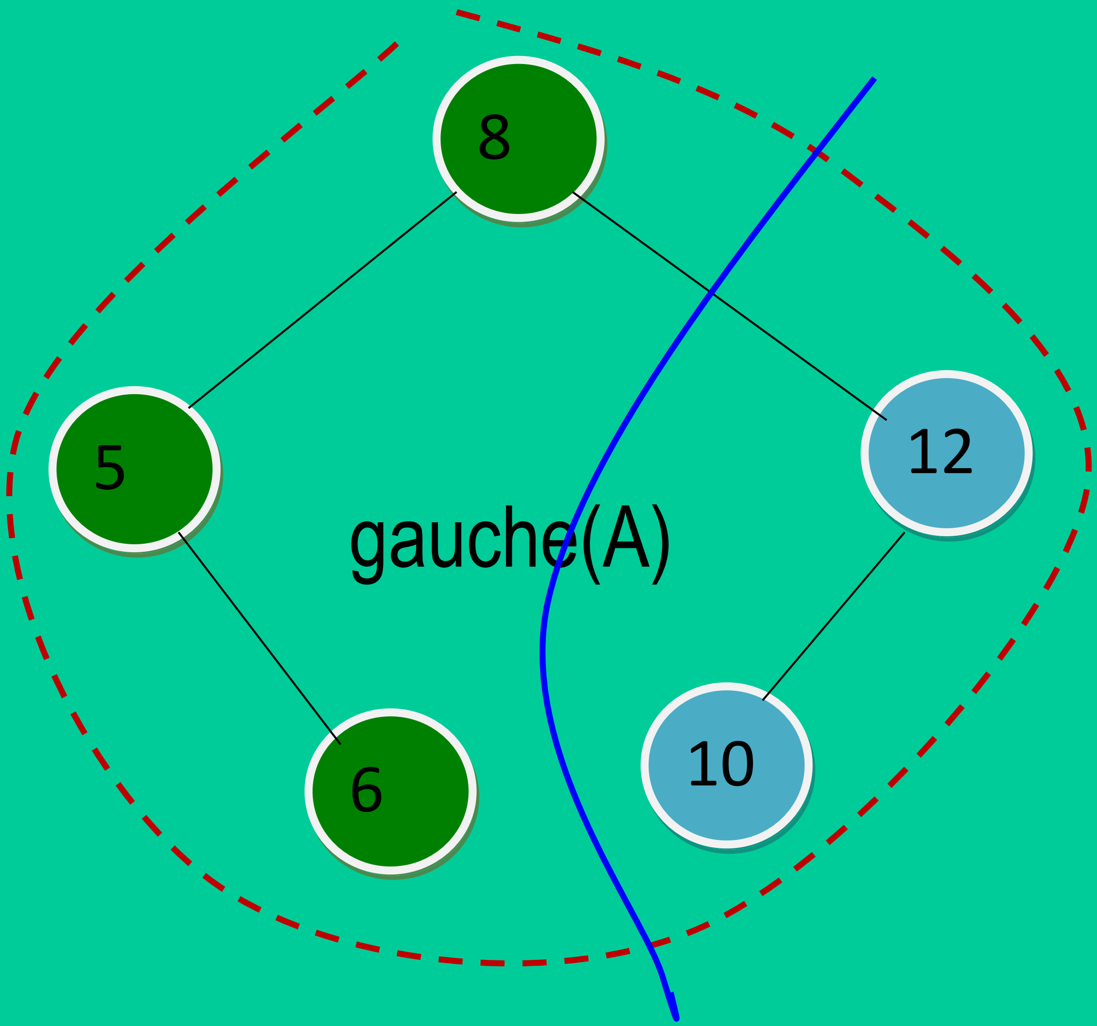
X< 15 : coupure à gauche

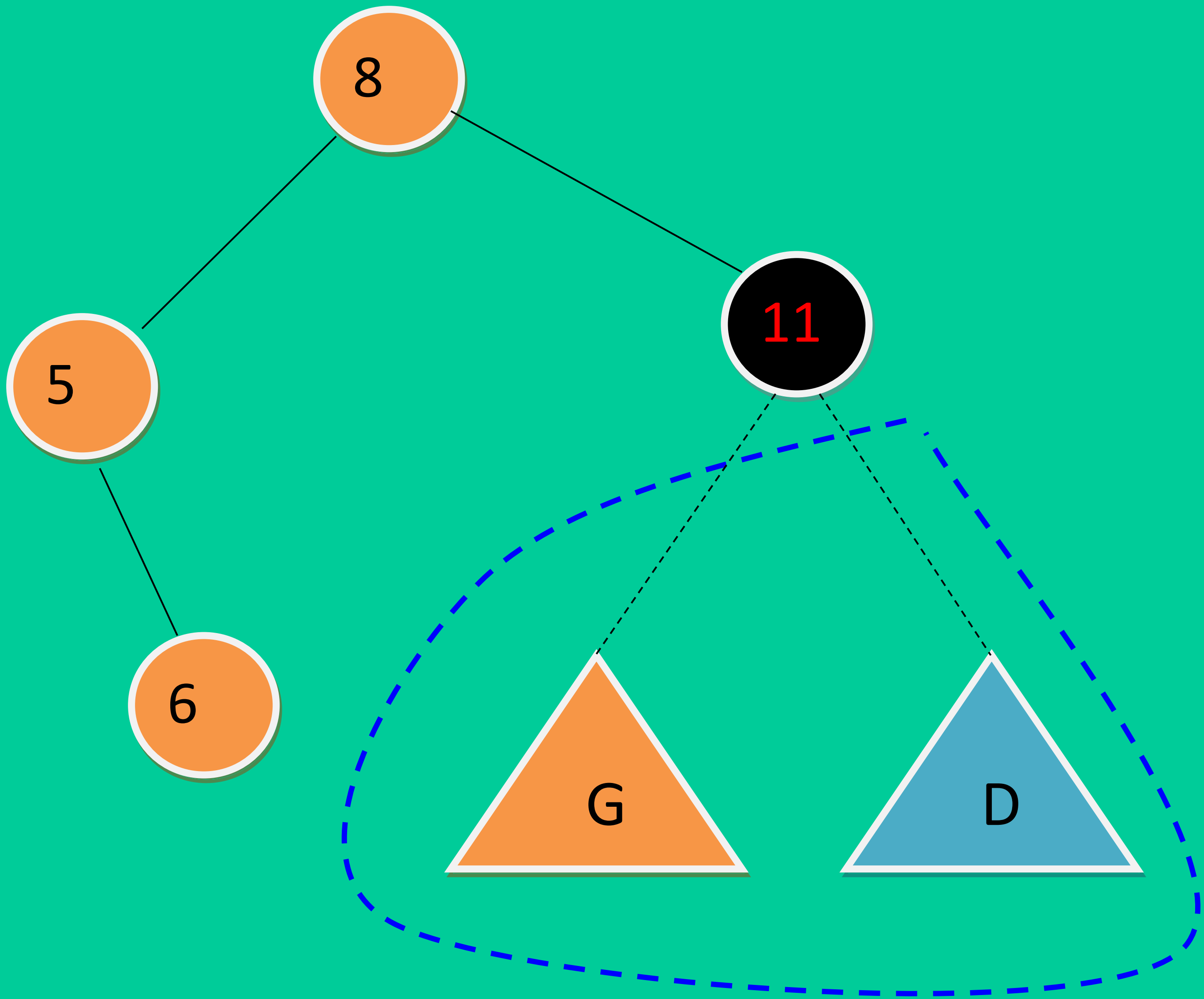


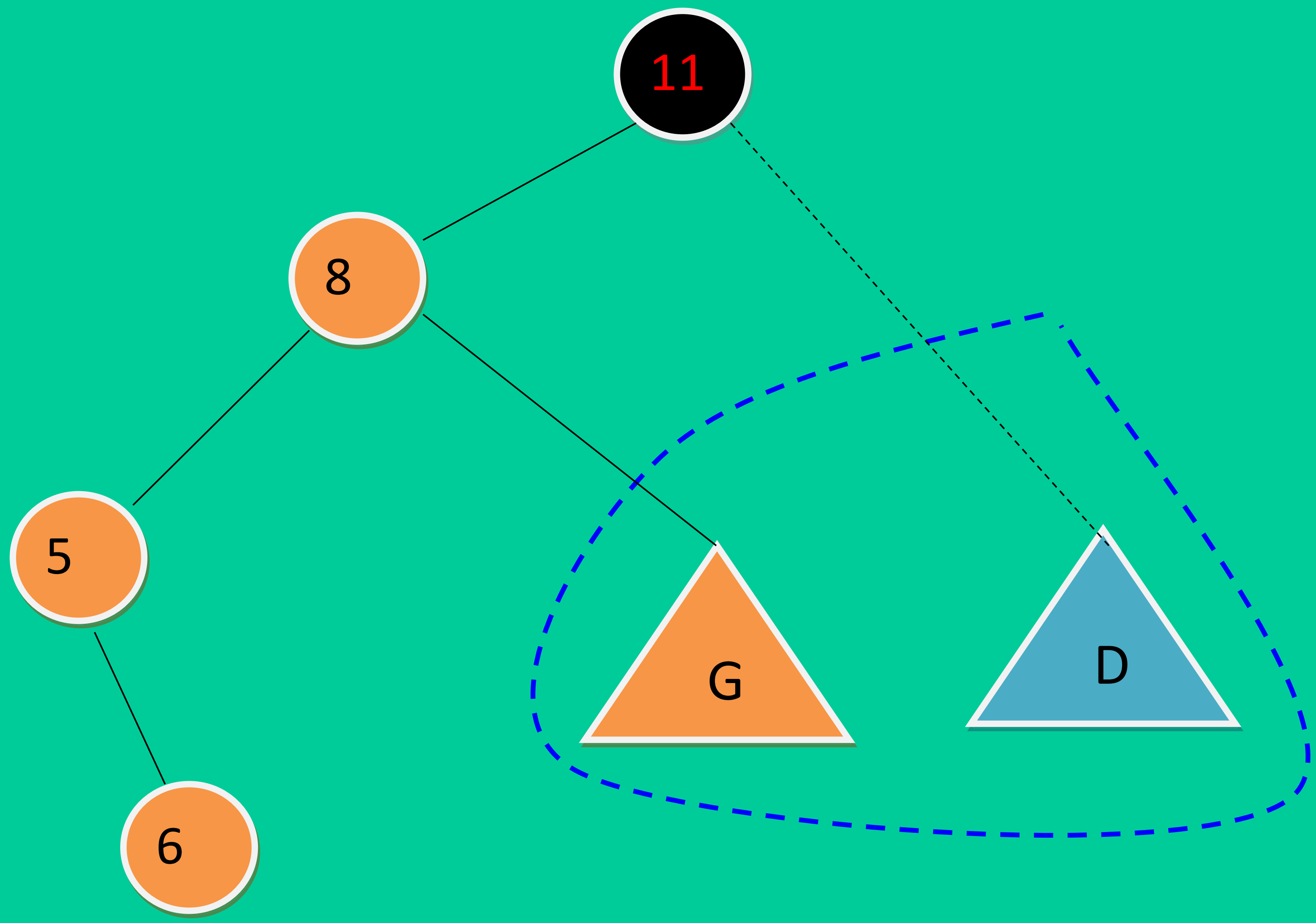

15
11
29
21
32
24
G
D
gauche (A)
droit (A)


11
15
29
21
32
24
G
D
gauche (A)
droit (A)

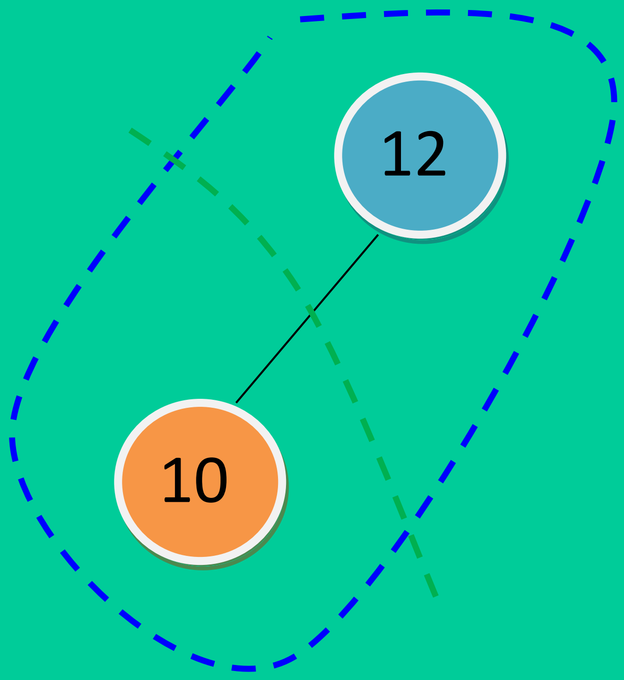
x =11 > 8 : coupure à droite

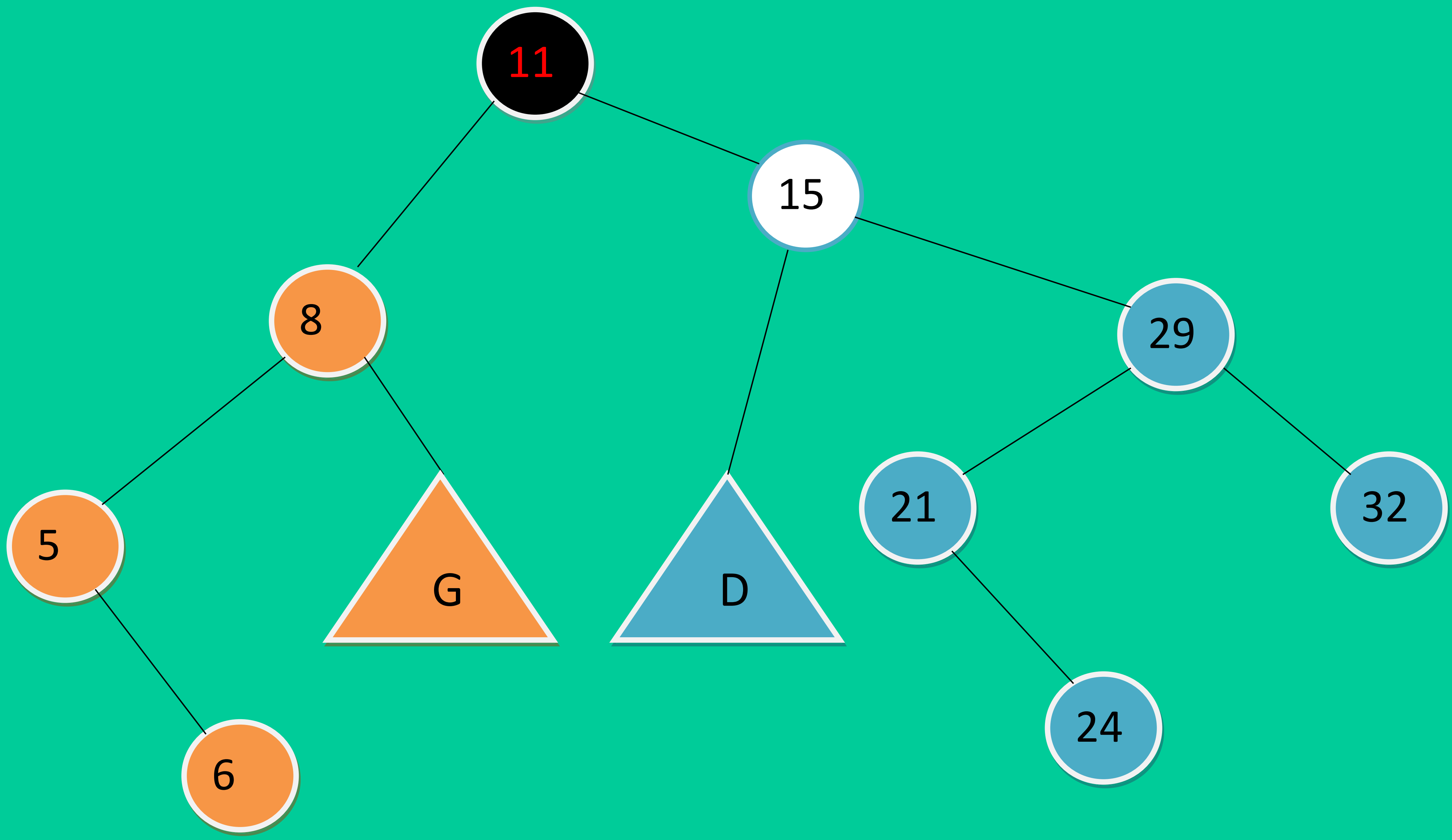


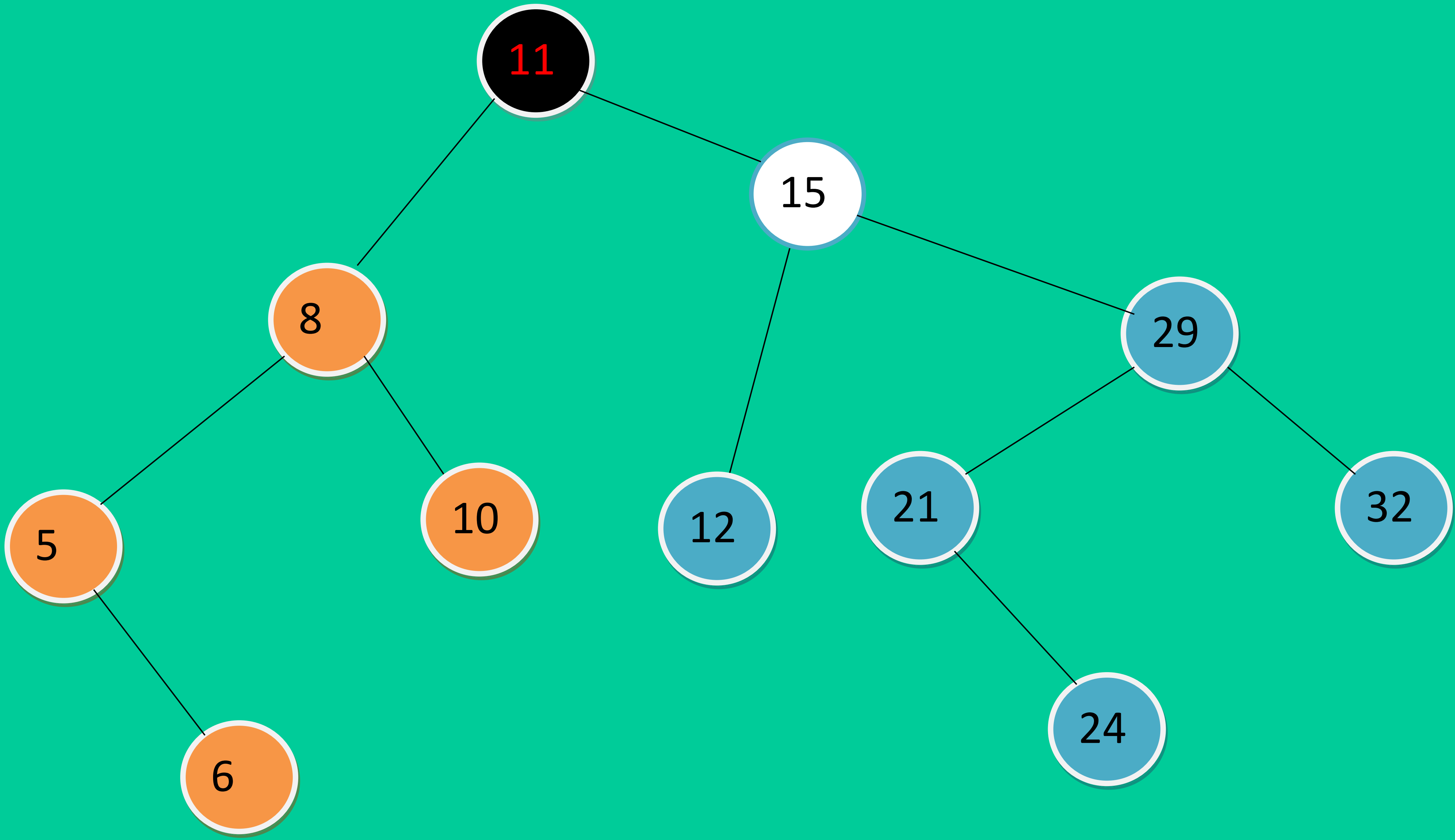

8
5
11
6
G
D


11
8
5
6
G
D

X= 11 < 12 :coupure à gauche




11
15
8
29
5
G
D
21
32
6
24


11
15
8
29
5
10
12
21
32
6
24

Algorithmes sur les ABR

1-Parcours en profondeur d'un ABR

On visite chacun des **n** nœuds de l'arbre une et une seule fois: **ordre infixe**.

L'algorithme s'exécute toujours en un temps linéaire en **n** s'il y a **n** éléments composant l'ABR : Il n'y a **pas de pire des cas**.

2-Construction d'un ABR à partir d'une liste

La construction est le résultat de l'application de l'algorithme d'insertion de chaque élément de la liste dans un ABR

L'ABR devra comporter au **maximum** autant de nœuds que d'éléments **n** à insérer.

a) Dans **le meilleur cas** et en moyenne, on élimine à chaque itération la moitié des nœuds de l'arbre.

Combien de fois doit-on diviser **n** par deux jusqu'à tomber sur 1 ?

Il s'agit de la fonction logarithme (ici en base 2).

L'algorithme d'insertion est donc en **O(log(n))**

Pour insérer chacun des **n** élément de la liste dans l'arbre, on a donc logiquement un temps en **O(nlog(n)).**

b) Dans **le pire des cas**, l'arbre est totalement balancé d'un côté et on obtient une liste.

Alors, l'insertion impose donc un parcours de tous les éléments et une complexité pour l'insertion en **$O(n)$.**

Globalement, la construction de l'ABR se fera donc en **$O(n^2)$,** appelée complexité **quadratique**.

3-Le tri par ABR

Le tri par ABR est le résultat de l’application successive:

1- de l'algorithme de construction d'un ABR à partir d'une liste:

2- puis de l'algorithme du parcours en profondeur **infixe** pour récupérer les valeurs dans un ordre trié :