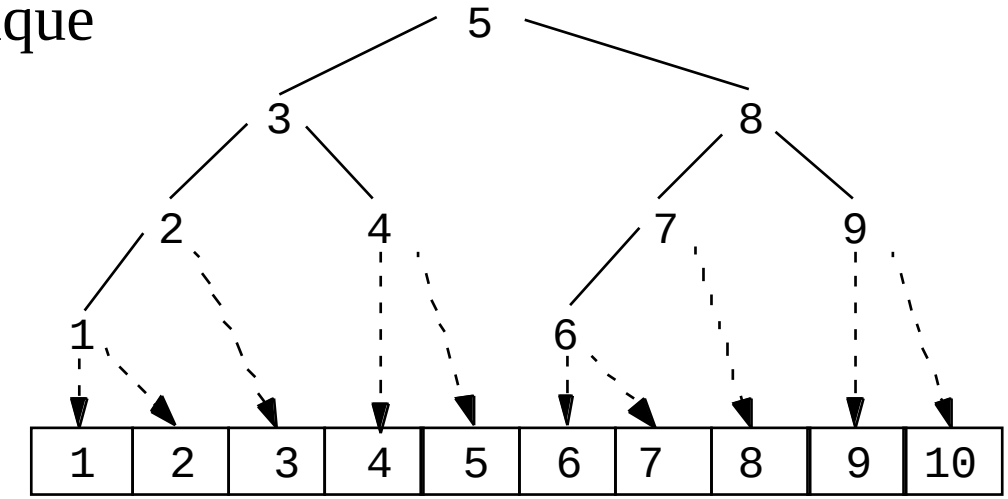
Arbres binaires de recherche

arbre des comparaisons

- recherche dichotomique
  - l'arbre est recalculé à chaque recherche



- conserver la structure d'arbre au lieu de la reconstruire
- arbre binaire de recherche:
  - <o, a, b> => max(a) ≤ la_clé(o) ≤ min(b)

le type arbre-rech

type arbre-rech

opérations

<_, _, _> : nœud × arbre-rech × arbre-rech /→ arbre-rech

max : arbre-rech /→ clé

min : arbre-rech /→ clé

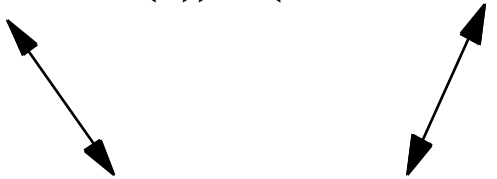
_ _ : clé × arbre-rech → booléen

préconditions

<o, g, d>: (g ≠ Ø max(g) ≤ la_clé(o)) (d ≠ Ø min(d) ≥ la_clé(o))

max(a): a ≠ Ø

min (a): a ≠ Ø



utilisation conjointe
axiome et précondition

sémantique

max(<o, g, d>) = **si** d = Ø **alors** la_clé(o) **sinon** max(d) **fsi**

min(<o, g, d>) = **si** g = Ø **alors** la_clé(o) **sinon** min(g) **fsi**

c a = **si** a = Ø **alors** faux

sinon soit a = <o, g, d> ; c = la_clé(o) c g c d **fsi**

Implantation complète (Ada)


```
generic
    type Element is private;
    type Cle is private;
    with function La_Cle(E: in Element) return Cle is <>;
    with function "<"(U, V: in Cle) return Boolean is <>;
package Arbres_Binaires.De_Recherche is

    type Arbre_Rech is new Arbre with null record;
    procedure Positionner (L_Arbre: in out Arbre_Rech ; Sur: in Cle);
    function Contenu_Courant (L_Arbre: in Arbre_Rech) return Element;
    procedure Changer_Courant (L_Arbre : in out Arbre_Rech; Val : in Element);
    procedure Ajouter (Dans : in out Arbre_Rech; Val : in Element);
private
    type Noeud_Rech is new Noeud with record  Contenu : Element; end record;
    procedure Ajouter_Feuille(Dans:in out Arbre_Rech;La_Feuille:in Pt_Noeud);
end Arbres_Binaires.De_Recherche;
```

visibilité sur la partie privée du père

réutilisation des arbres binaires

extension sur le nouveau type

Implantation complète (Java)


```
package bibSDJava.Les_Types_Generiques;
public interface Comparable extends Cloneable1 {
   boolean Inferieur (Object obj) throws Erreur_Typage; }
 public interface Avec_Cle extends Cloneable1 {
   Comparable La_Cle (); }
package bibSDJava.Les_Arbres_Binaires;
class Noeud_Recherche extends Noeud { Avec_Cle Contenu; }
public class Arbre_De_Recherche extends Arbre_Binaire {
   protected Class Type_Element;
   protected Class Type_Cle;
   public Arbre_De_Recherche (Avec_Cle Specimen) throws Erreur_Typage {}
   public void Positionner (Comparable Sur)
                throws Cle_Inconnue, Limite_Implantation, Erreur_Typage {}
   public Object Contenu_Courant () throws Erreur_Specification {}
}
```

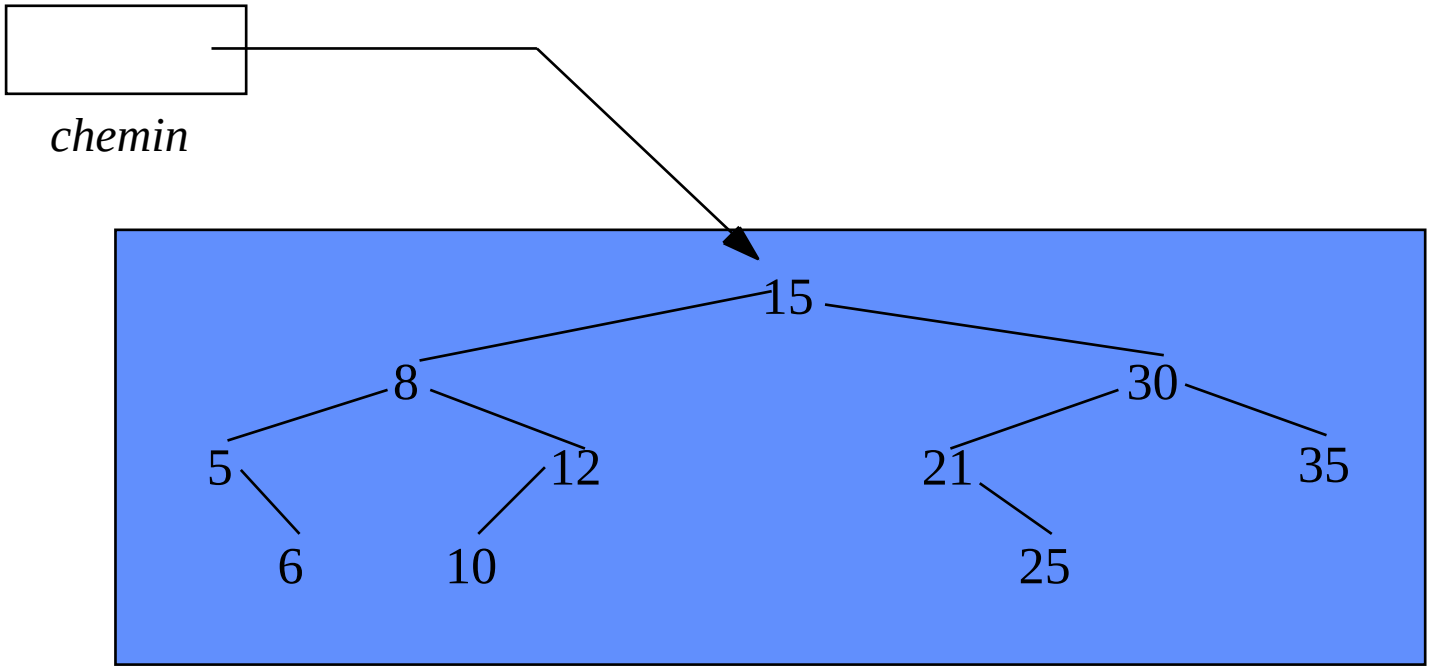
propriétés attendues sur les valeurs des nœuds

extension des nœuds

extension des opérations

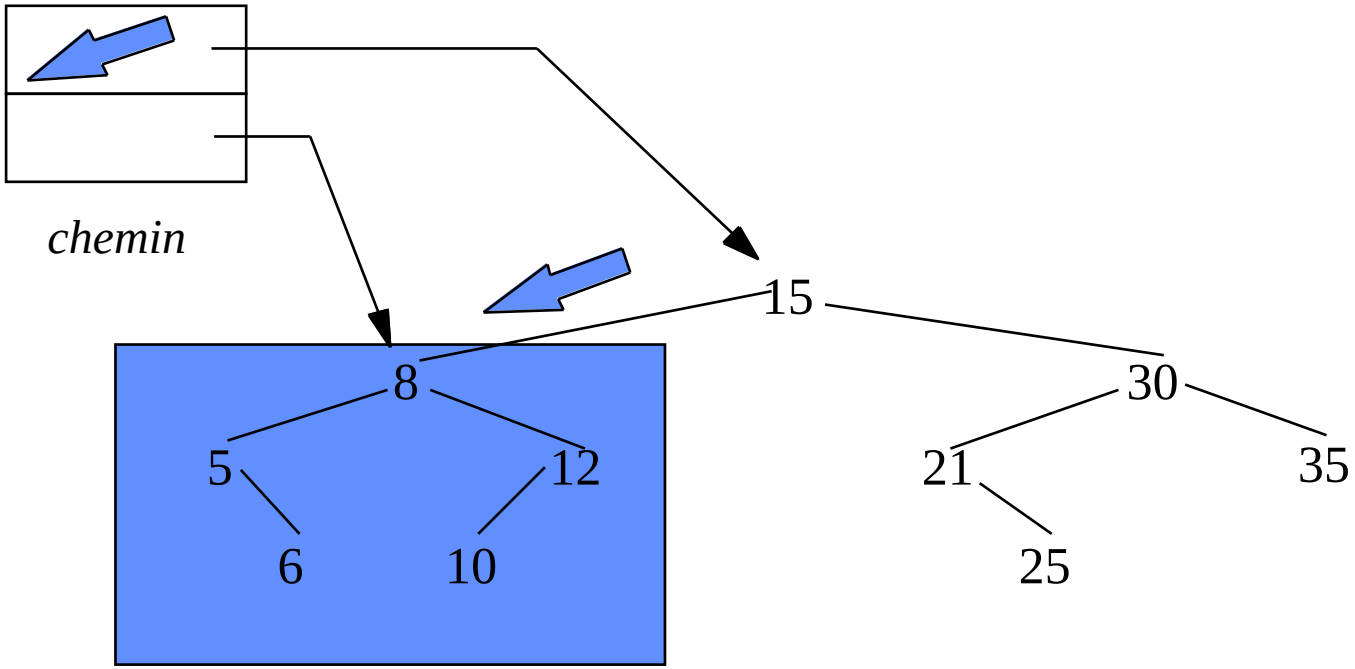
Exemple de recherche

- Soit à rechercher 12 dans l'arbre suivant



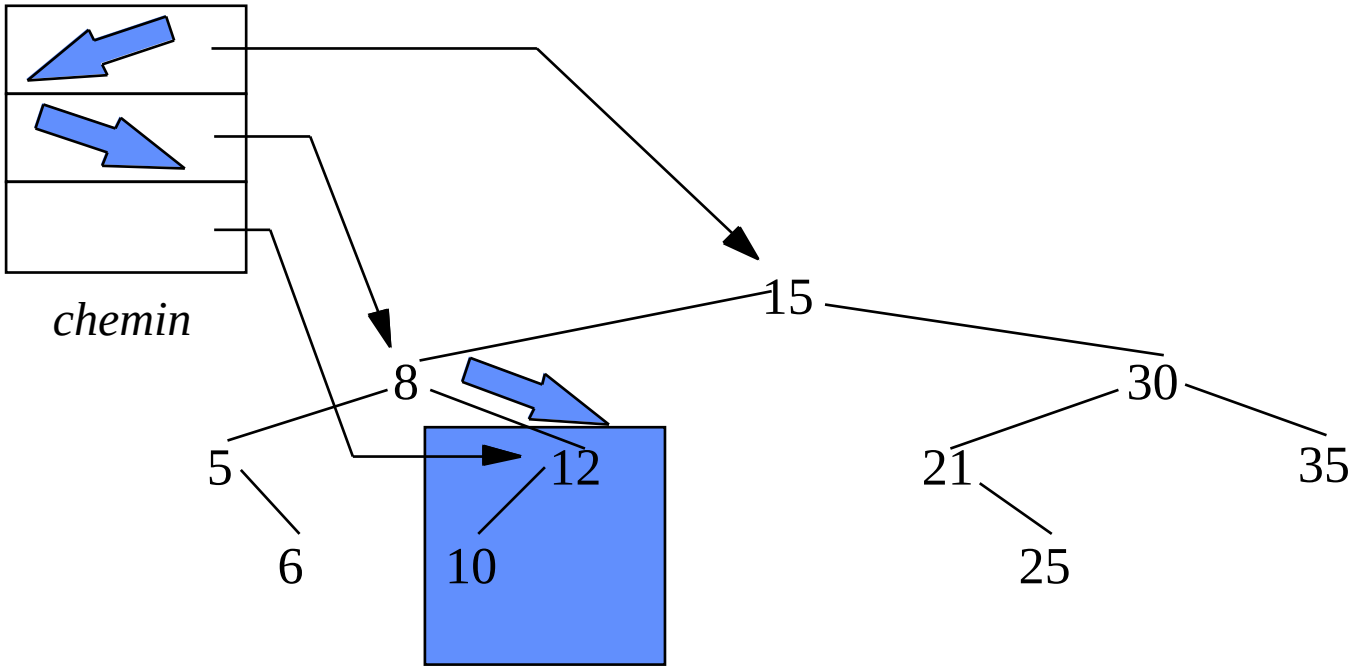
Exemple de recherche

- 12 est plus petit que 15



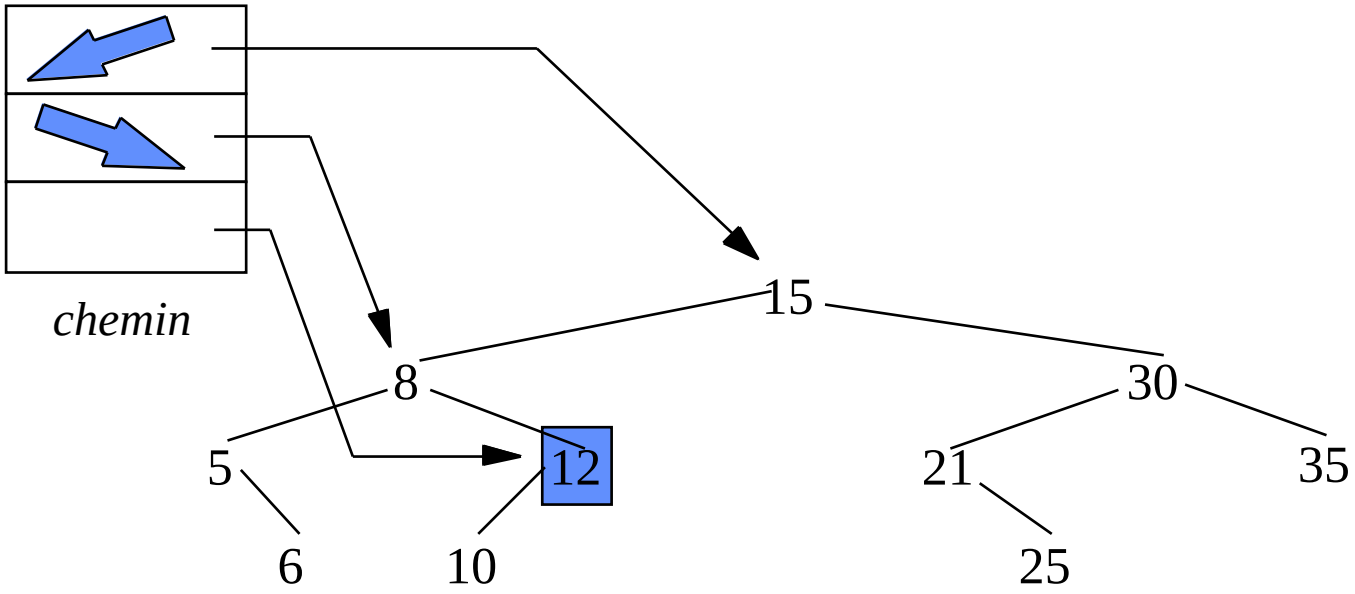
Exemple de recherche

- 12 est plus grand que 8



Exemple de recherche

- 12 est trouvé



Opération de recherche

- Spécification

  extension type arbre-rech

  opérations **préconditions**

  recherche : clé × arbre-rech /→ nœud c a

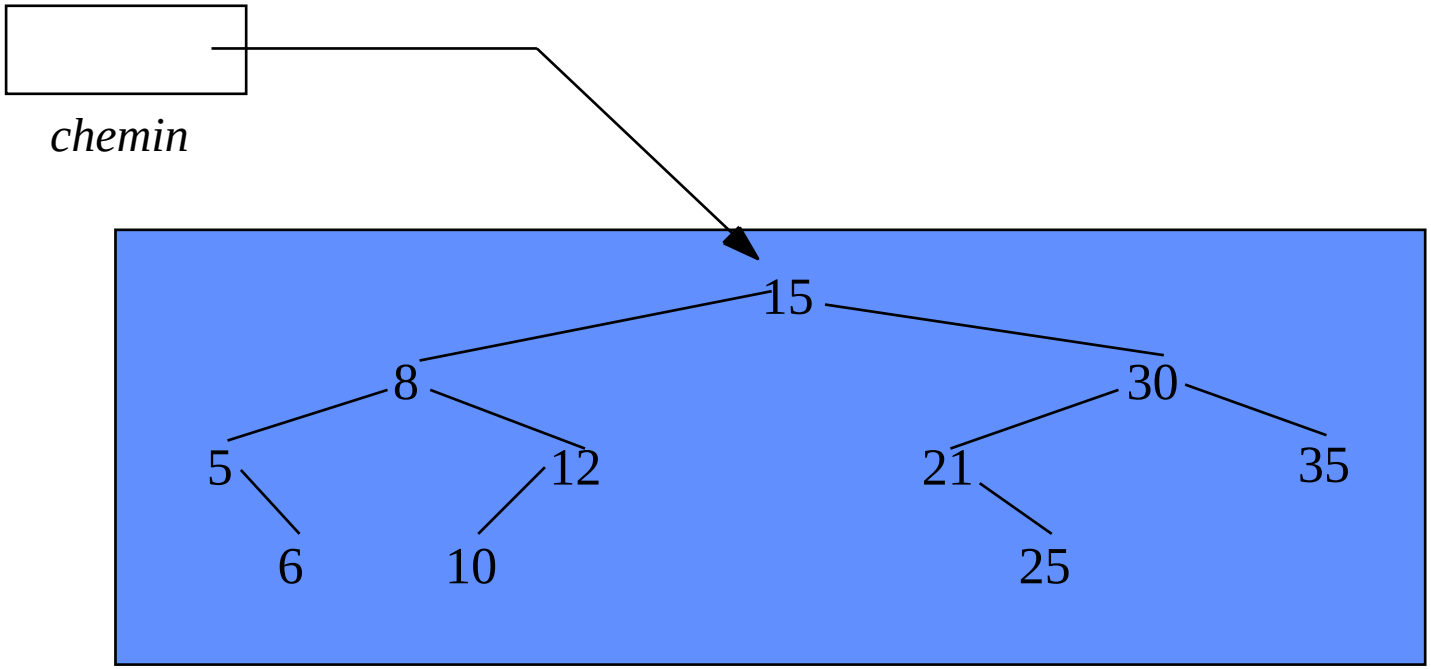
  sémantique

  recherche(c, a) = **si** c = la_clé(racine(a)) **alors** racine(a)
  sinsi c < la_clé(racine(a)) **alors** recherche(c, g(a))
  sinon recherche(c, d(a)) **fsi**

- complexité: en moyenne PC(a), au pire h(a)
- utilisation de l'implantation des arbres binaires
- Note: on construit le chemin jusqu'au nœud cherché
  - implantation : **procedure** Positionner (L_Arbre : **in out** Arbre_Rech;
    Sur : **in** Cle)

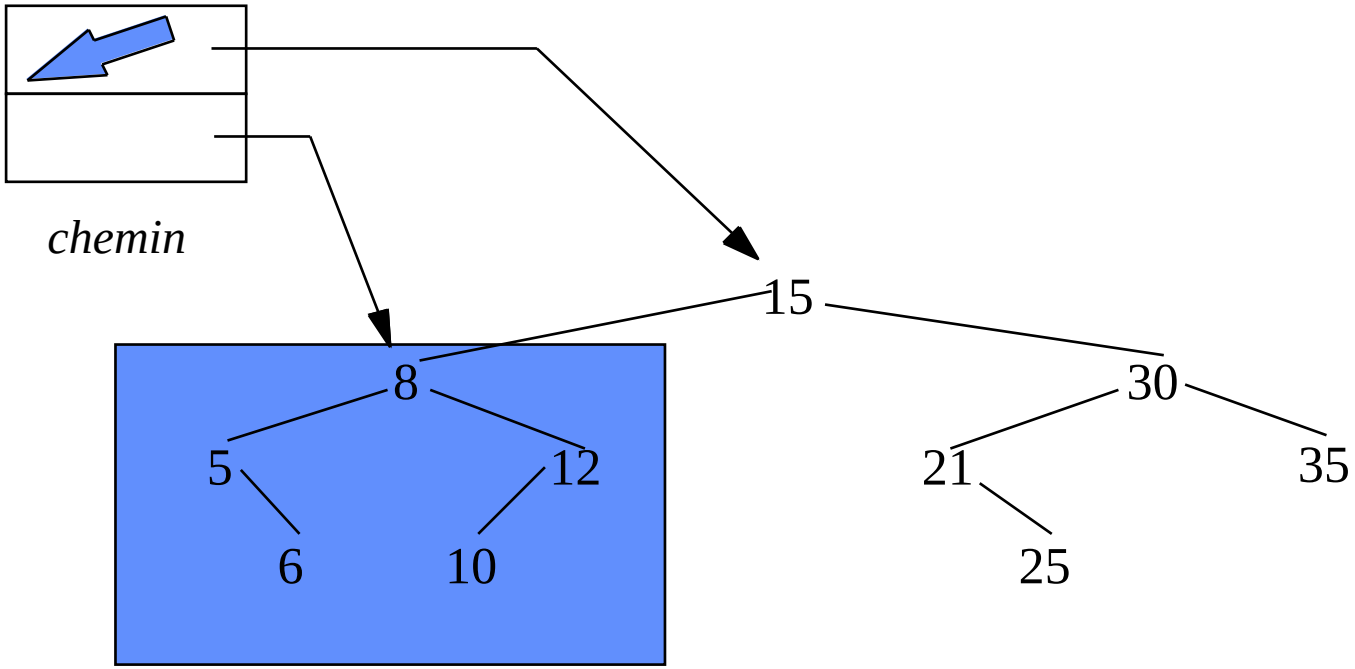
Exemple d'adjonction

- Soit à ajouter 14 dans l'arbre suivant



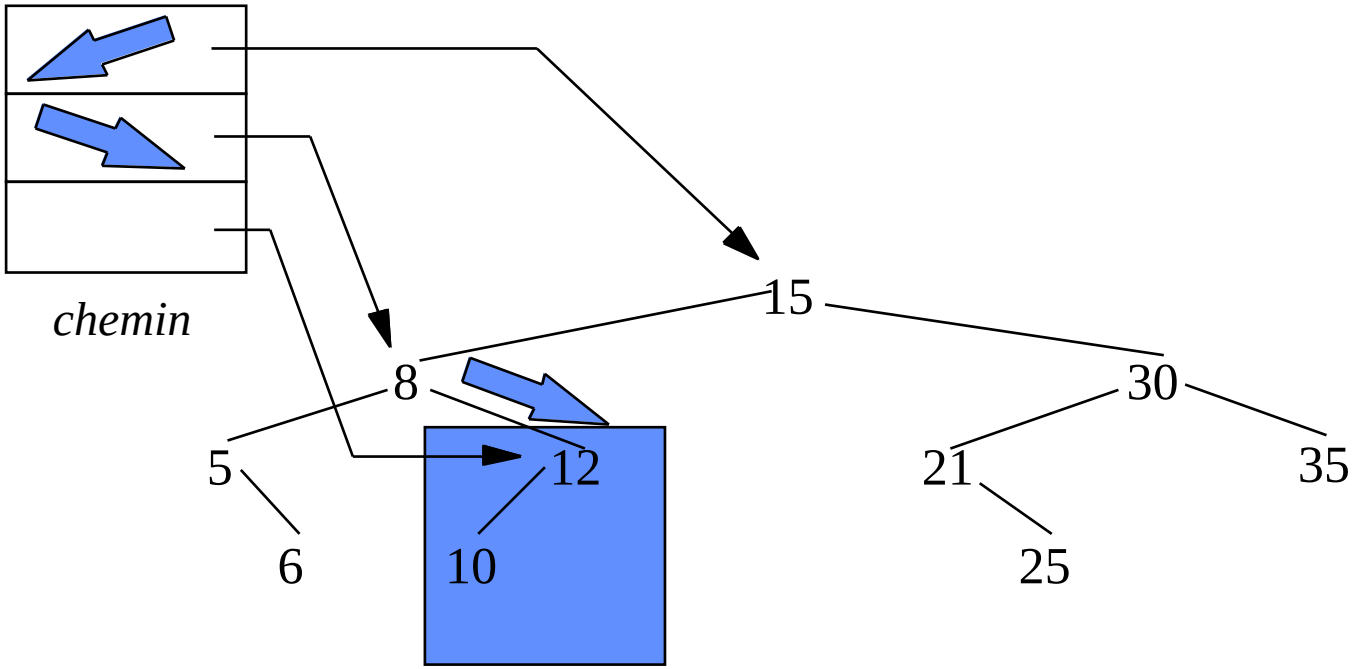
Exemple d'adjonction

- 14 est plus petit que 15



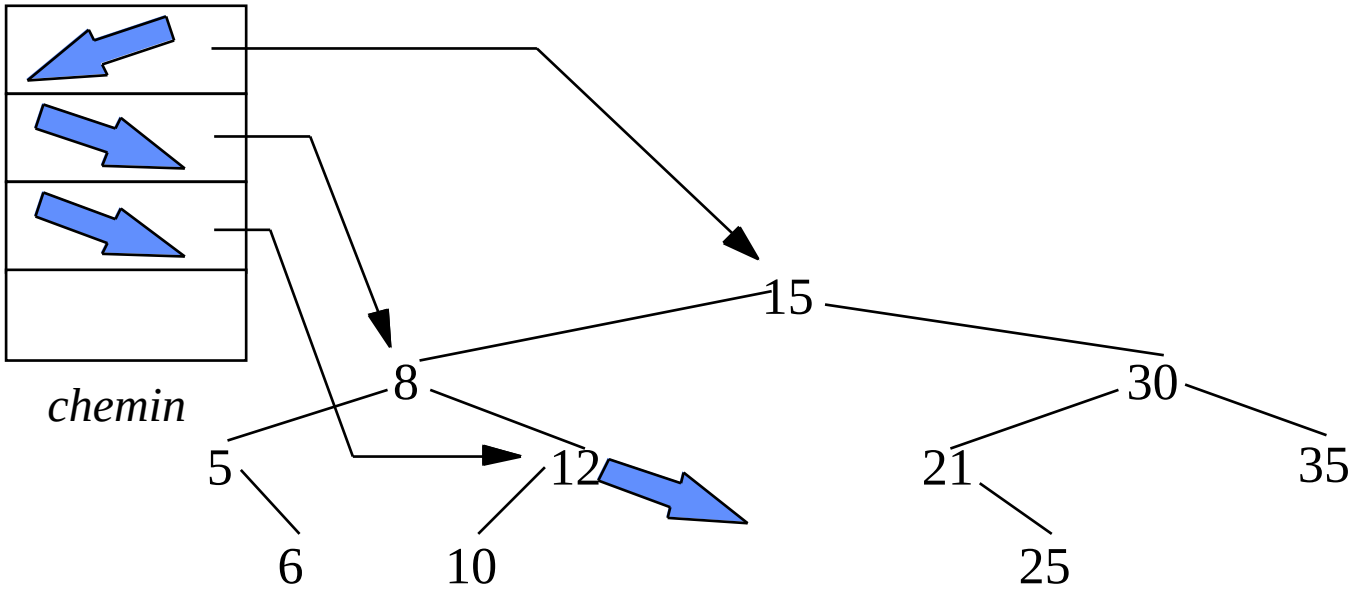
Exemple d'adjonction

- 14 est plus grand que 8



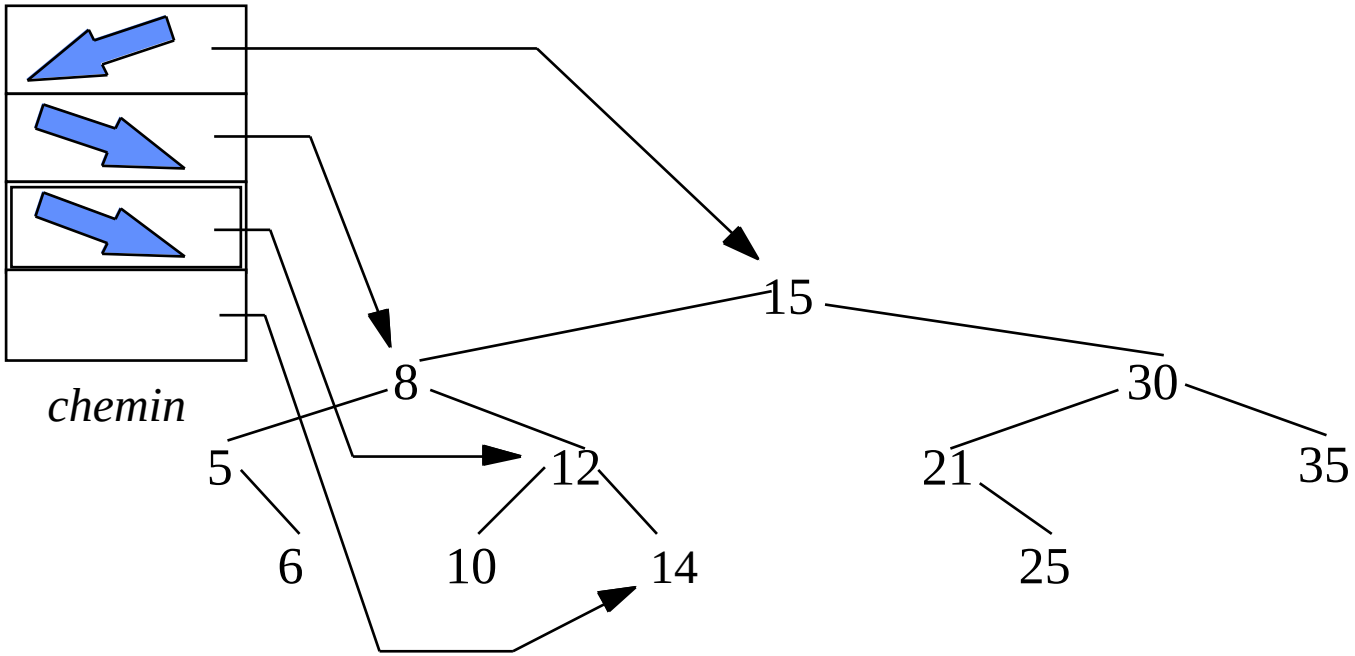
Exemple d'adjonction

- 14 est plus grand que 12 => repère un arbre vide



Exemple d'adjonction

- remplacement arbre vide par le nœud 14



Adjonction d'une feuille (1)

- appliquer la recherche =>arbre vide remplacé par la feuille

extension type arbre-rech

opérations **préconditions**

ajouter-f : nœud × arbre-rech /→ arbre-rech ¬ (la_clé(o) a)

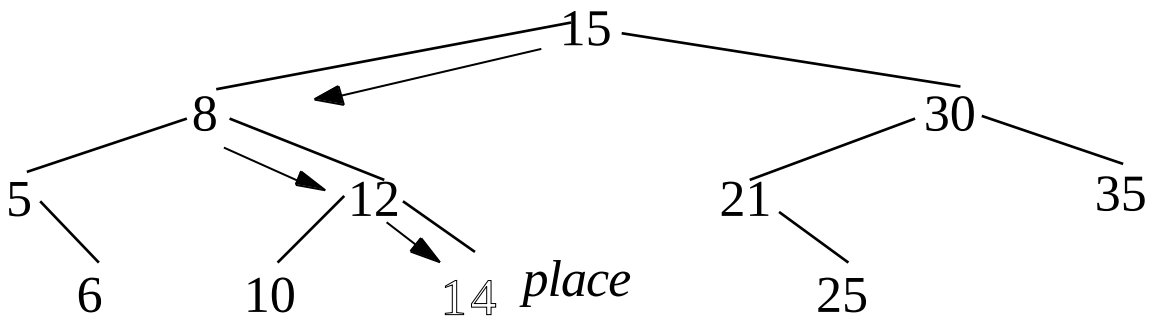
sémantique

ajouter-f(o, a) = **si** a = Ø **alors** <o, Ø, Ø>

sinon soit a = <o', g, d>;

si la_clé(o) < la_clé(o') **alors** <o', ajouter-f(o, g), d>

sinon <o', g, ajouter-f(o, d)> **fsi** **fsi**



Adjonction d'une feuille (2)


```
procedure Ajouter (Dans : in out Arbre_Rech; Val : in Element) is
    Cval : Cle := La_Cle (Val);
    Le_Noeud: Pt_Noeud:= L_Arbre.Chemin(1).Le_Noeud;
begin
    L_Arbre.Sommet := 1;
    while Le_Noeud /= null
    and then Cval /= La_Cle (Le_Noeud.Contenu) loop
        Descendre(L_Arbre, A_Gauche=> Cval < La_Cle(Le_Noeud.Contenu));
        Le_Noeud := L_Arbre.Chemin(L_Arbre.Sommet).Le_Noeud;
    end loop;
    if Le_Noeud /= null then raise Erreur_Specification; end if;
    Raccrocher(Dans, new Noeud_Rech'(Contenu => Val, others => null));
end Ajouter_Feuille;
```

recherche

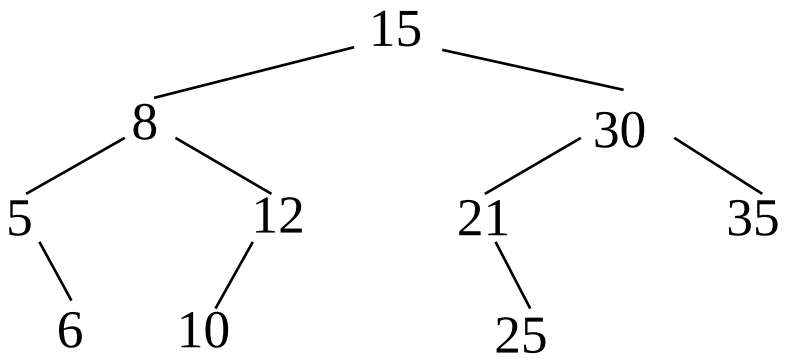
adjonction

mise au bout du chemin courant

complexité: en moyenne PCE(a), au pire h(a)

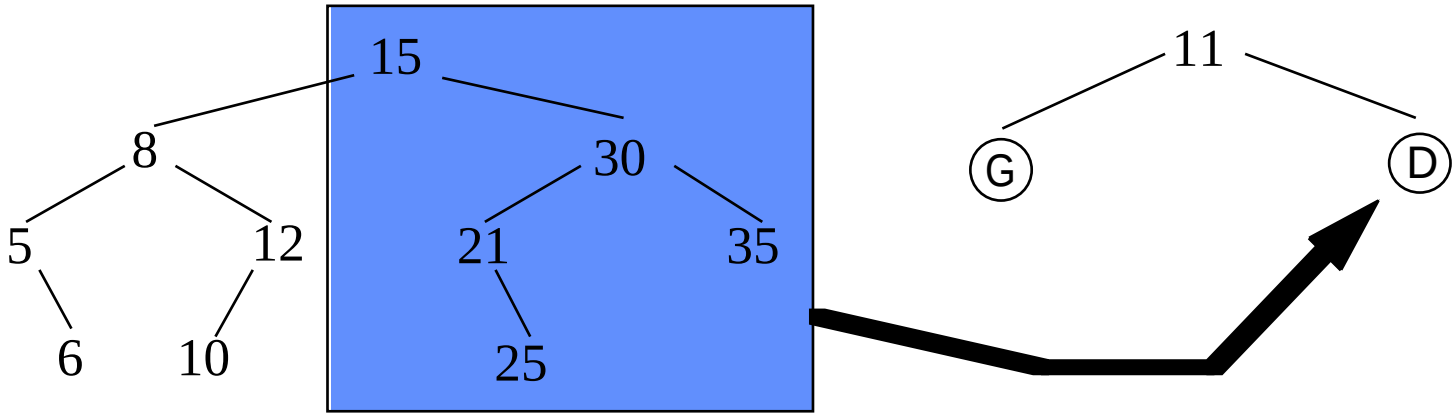
Exemple d'adjonction à la racine

- Soit à ajouter 11 à l'arbre suivant
- reconstruire l'arbre autour du nœud en tant que racine



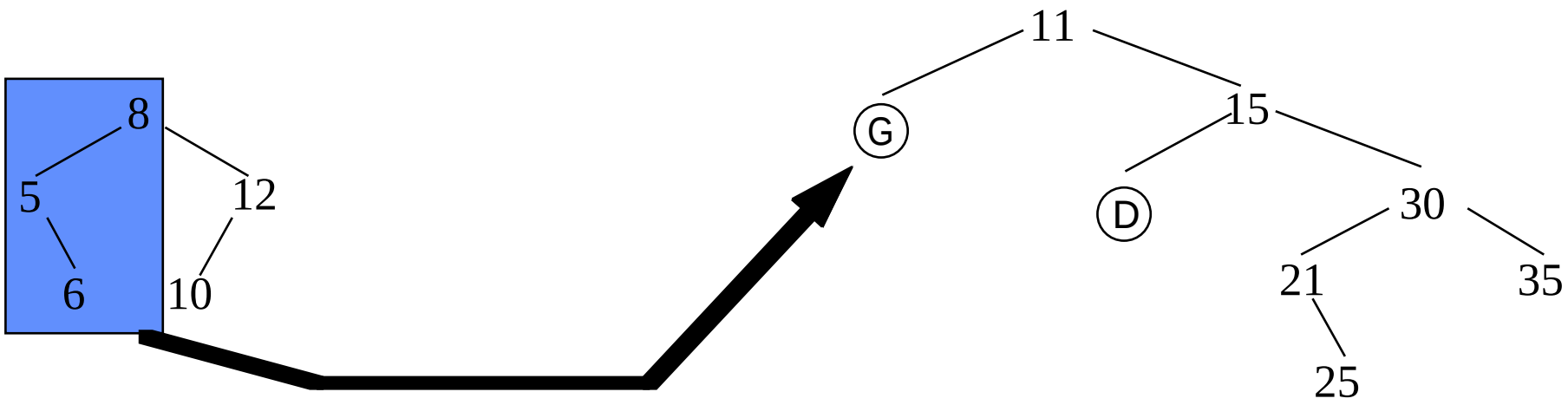
Exemple d'adjonction à la racine

- 15 est plus grand que 11 donc à droite de 11
- tous ceux du sous-arbre droit de 15 sont plus grands que 15, donc plus grands que 11
- ceux qui restent sont plus petits que 15



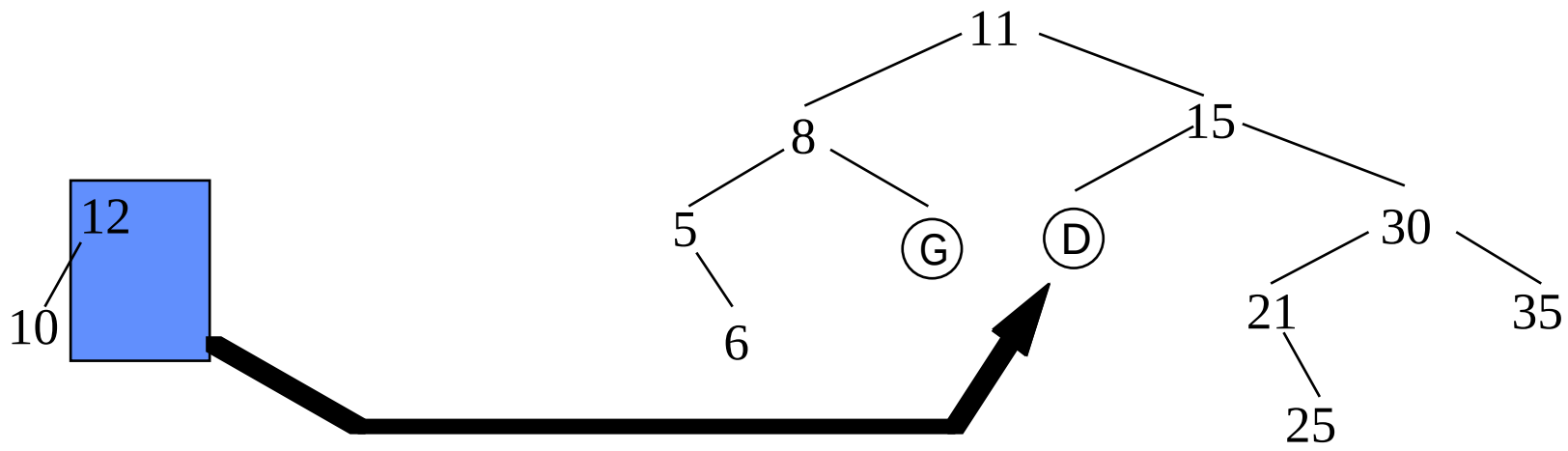
Exemple d'adjonction à la racine

- 8 est plus petit que 11
- tous ceux du sous-arbre gauche de 8 sont plus petits que 8, donc plus petits que 11
- ceux qui restent sont plus petits que 15 et plus grands que 8



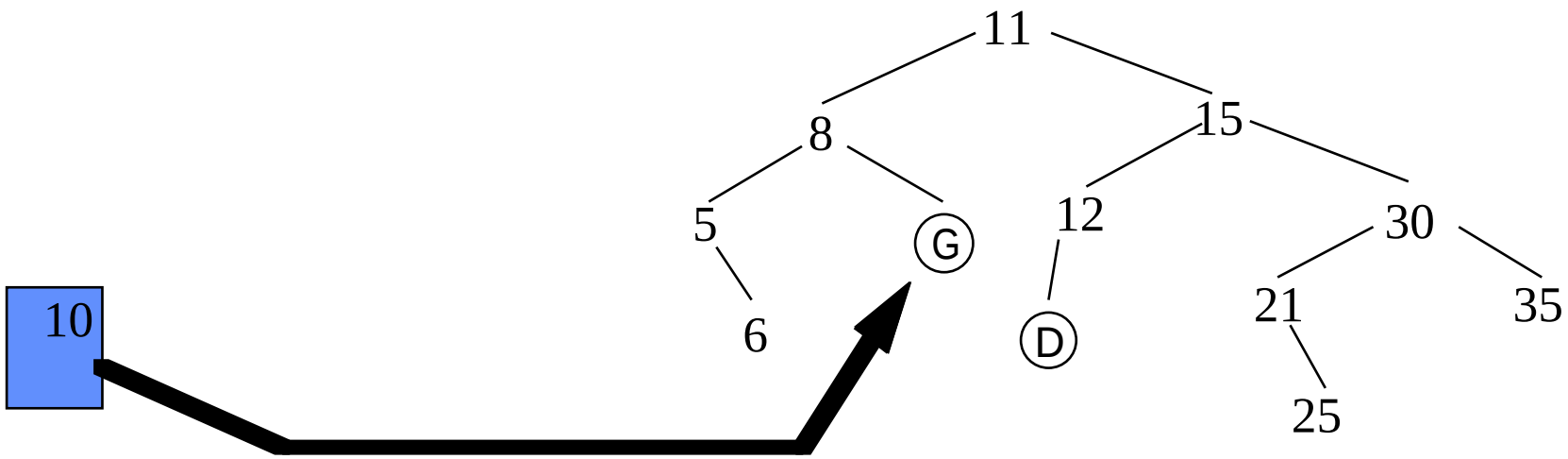
Exemple d'adjonction à la racine

- 12 est plus grand que 11 (et plus petit que 15)
- ceux de son sous-arbre droit sont plus grands que 12, donc que 11 et plus petits que 15
- ceux qui restent sont plus grands que 8 et plus petits que 12



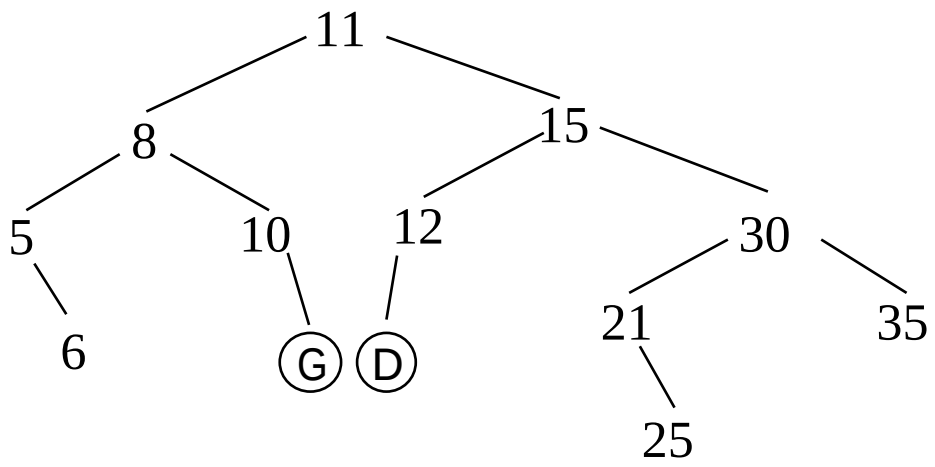
Exemple d'adjonction à la racine

- 10 est plus petit que 11 (et plus grand que 8)
- ceux de son sous-arbre gauche sont plus petits que 10, donc que 11, et plus grands que 8
- ceux qui restent sont plus grands que 10 et plus petits que 12



Exemple d'adjonction à la racine

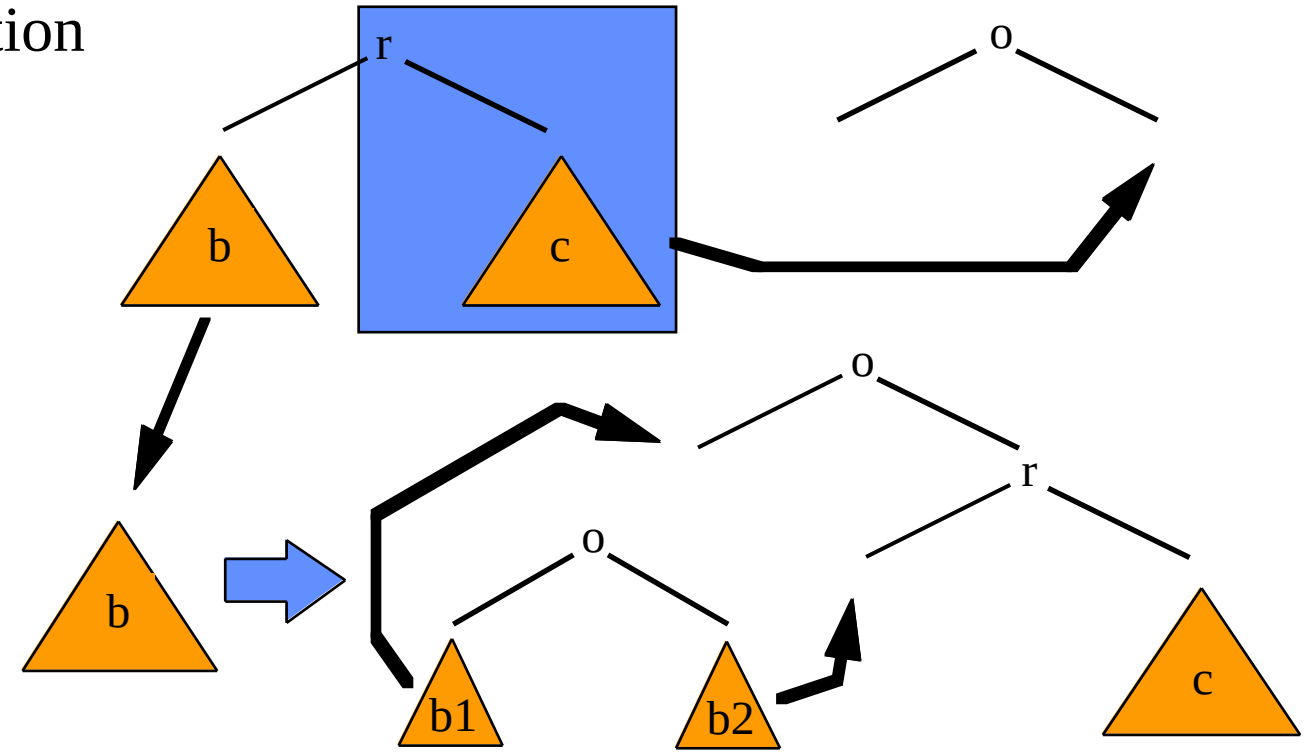
- l'arbre restant est vide: il n'y a pas de valeurs entre 8 et 12, les arbres G et D sont vides.



Adjonction à la racine (1)

- Généralisation

  cas o < r



ajout(o, <r, b, c>) = <o, g(ajout(o, b), <r, d(ajout(o, b)), c>>

Adjonction à la racine (2)

- spécification axiomatique

extension type arbre-rech

opérations

ajouter-r : nœud × arbre-rech /→ arbre-rech

préconditions

ajouter-r(o, a): ¬ (la_clé(o) a)

sémantique

ajouter-r(o, a) = **si** a = Ø **alors** <o, Ø, Ø>

sinon soit a = <o', g, d>;

si la_clé(o) < la_clé(o') **alors**

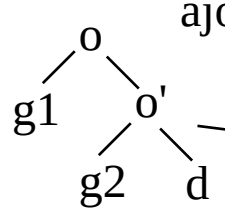
<o, g(ajouter-r(o, g)), < o', d(ajouter-r(o, g)), d>>

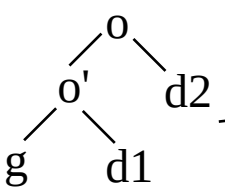
sinon

<o, <o', g, g(ajouter-r(o, d))>, d(ajouter-r(o, d))>

fsi

fsi





Adjonction à la racine (3)

- Implantation récursive

```
procedure Ajouter_Racine (Val : in Element; Racine : in out Pt_Noeud) is
   Nouveau:Pt_Noeud := new Noeud'(Contenu=>Val,others=>null);
   procedure Coupure (Courant: in Pt_Noeud; Gauche, Droite : out Pt_Noeud) is
   begin
      if Courant = null then Gauche := null; Droite := null;
      elsif La_Cle (Val) = La_Cle (Courant.Contenu) then
         raise Erreur_Specification;                    -- violation précondition
      elsif La_Cle (Val) < La_Cle (Courant.Contenu) then
         Droite := Courant; Coupure (Courant.Gauche, Gauche, Courant.Gauche);
      else Gauche := Courant; Coupure (Courant.Droite, Courant.Droite, Droite);
      end if;
   end Coupure;
begin Coupure (Racine, Nouveau.Gauche, Nouveau.Droite); Racine := Nouveau;
end Ajouter_Racine;
```

Adjonction à la racine (4)

- récursivité de coupure n'est pas terminale:

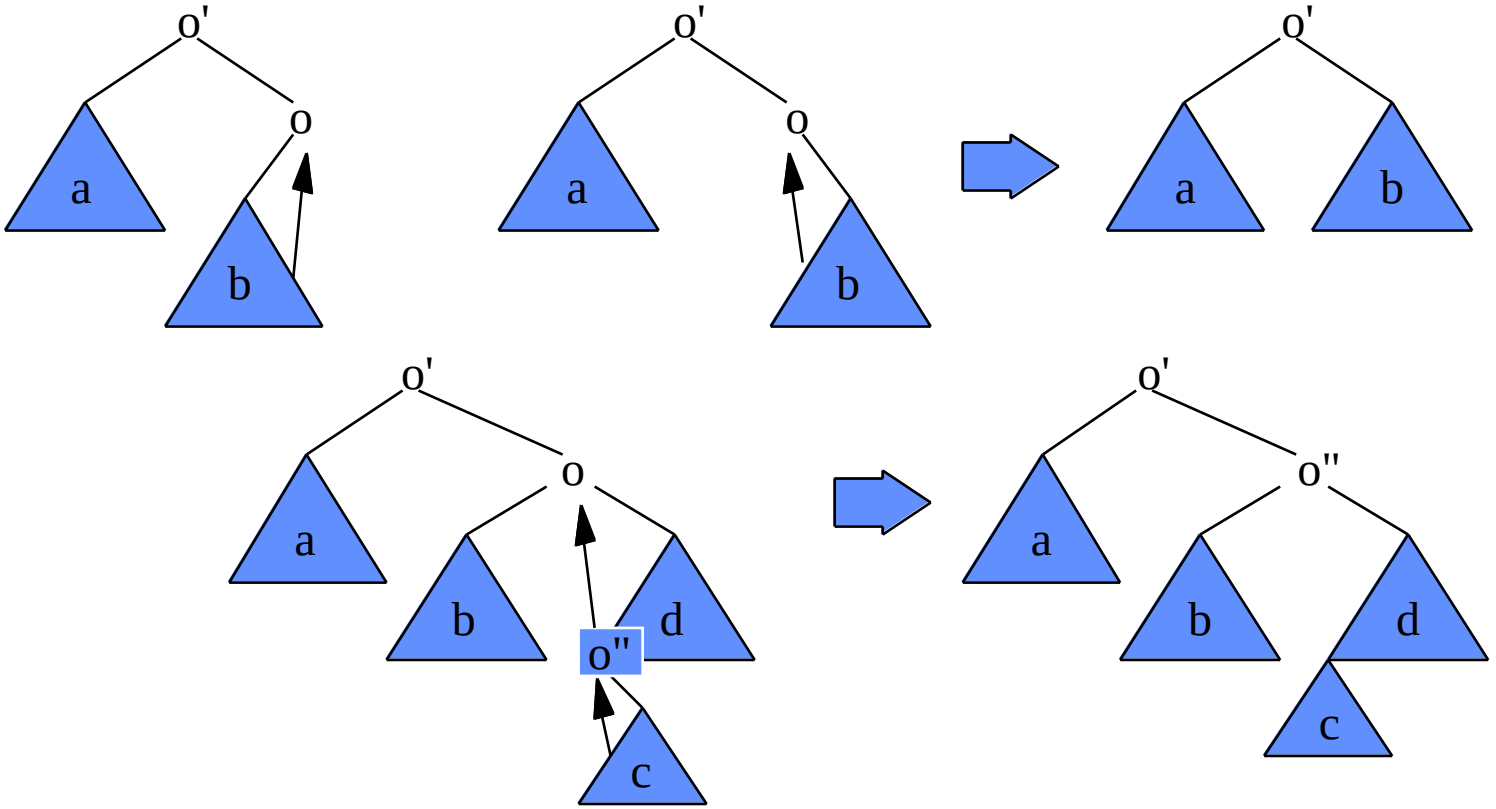
  procedure Coupure (Courant: **in** Pt_Noeud; Gauche, Droite : **out** Pt_Noeud)
  - ♦ les arguments Gauche et Droite sont modifiés au retour
  - ♦ si l'exception a lieu, les arguments ne sont pas modifiés
  - ♦ la pile a une profondeur proportionnelle à la hauteur de l'arbre
- dérécursification manuelle:
  - ♦ modification des pointeurs au fur et à mesure de la descente
  - ♦ en cas de levée de l'exception, reconstruire un arbre contenant les mêmes nœuds qu'au départ
- complexité: en moyenne PCE(a) au pire h(a)

Conclusion sur l'adjonction

- adjonction comme feuille ou comme racine, donne les mêmes complexités
  - en moyenne PCE(a)
  - au pire h(a)
- adjonction en feuille: les nœuds les plus anciens sont proches de la racine, et sont retrouvés plus vite
- adjonction en racine: les nœuds les plus récents sont proches de la racine, et sont retrouvés plus vite

Suppression (1)

- rechercher le nœud, puis supprimer selon 3 configurations:



Suppression (2)

extension type arbre-rech

opérations		**préconditions**
gauche	: arbre-rech /→ nœud	a ≠ Ø
sup-gauche	: arbre-rech /→ arbre-rech	a ≠ Ø
supprimer	: clé × arbre-rech /→ arbre-rech	c a

sémantique

gauche(<o, g, d>) = **si** g = Ø **alors** o **sinon** gauche(g) **fsi**

sup-gauche(<o, g, d>) = **si** g = Ø **alors** d **sinon** <o, sup-gauche(g), d> **fsi**

supprimer (c, <o, g, d>) = **si** c < la_clé(o) **alors** <o, supprimer(c, g), d>
sinsi c > la_clé(o) **alors** <o, g, supprimer(c, d)>
sinsi g = Ø **alors** d
sinsi d = Ø **alors** g
sinon <gauche(d), g, sup-gauche(d)> **fsi**

- complexité: en moyenne PCE(a) au pire h(a)

Conclusion

- recherche adjonction et suppression ont une complexité au pire en h(a), avec $\lfloor \log_2 n \rfloor \leq h(a) \leq n - 1$
- problème: moyenne des profondeurs des arbres de n nœuds lorsqu'on ajoute aléatoirement les nœuds?

  réponse: 2 Log n

  ⇒ *complexité moyenne d'ordre logarithmique*

- si on refuse les cas défavorables: arbres H-équilibrés