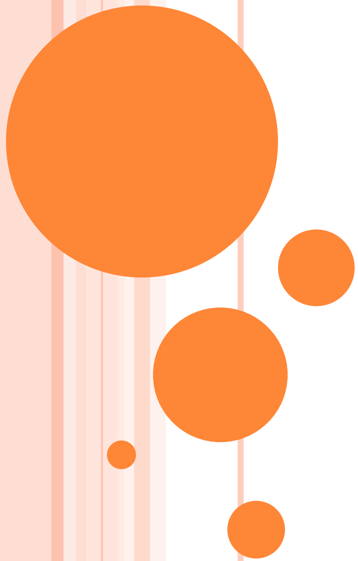
Tris en Prolog



Tris par sélection

Tri par insertion

Tri par fusion

TRI : QUEL EST LE PROBLÈME À RÉSOUDRE ?

- Soit une liste de nombres : [5,2,14,1,6]
- On veut construire une liste triée de ces éléments : [1,2,5,6,14]

ALGORITHMES DE TRI

- Tris par sélection du minimum
  - tri-minimum
  - tri-bulles
- Tri par insertion
- Tri par fusion
- Tri rapide
- Tri par tas

PRINCIPE DES TRIS PAR SÉLECTION

- On cherche le minimum de la liste à trier
- On le met au début de la liste qu'on construit
- Puis on recommence avec la liste à trier privée du minimum

EXERCICE : TRI DU MINIMUM

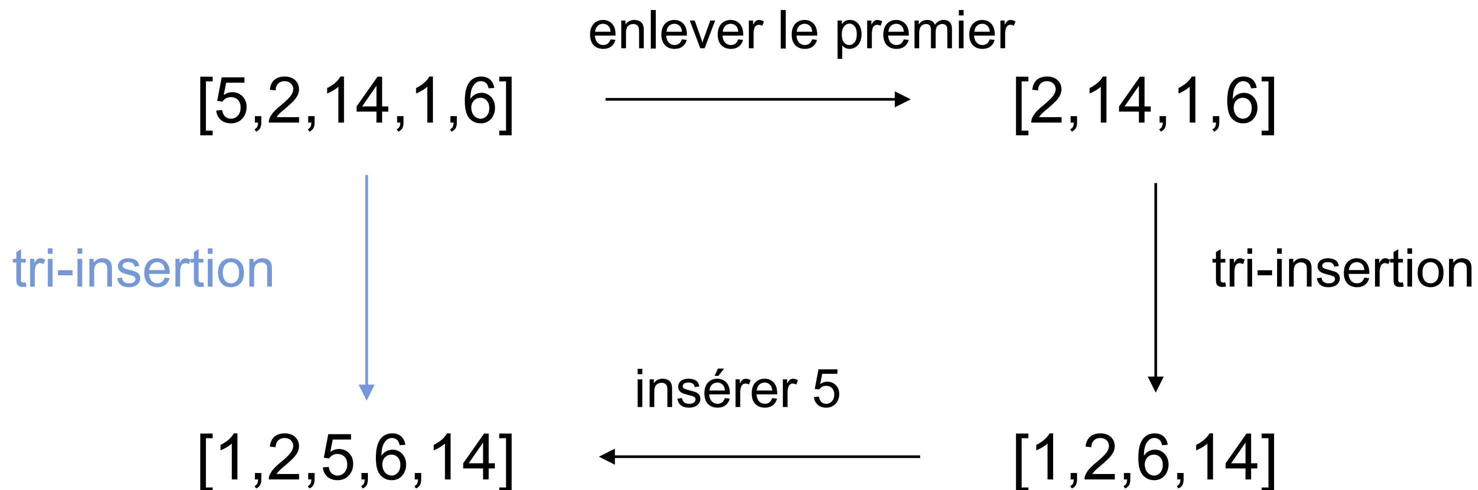
- Définir le prédicat minimum(L,X), qui calcule le minimum X de la liste de nombres L.
- Définir le prédicat enleve(X,L1,L2), qui construit la liste L2, qui est la liste L1 à laquelle on a enlevé la première occurrence de l'élément X.
- Utiliser les deux prédicats précédents pour définir le prédicat tri_min(L,Lt), qui construit la liste triée Lt à partir de la liste de nombres L.

Tri bulles (TP)

- Le tri bulles applique le principe du tri par sélection, mais utilise un seul prédicat bulle, qui sélectionne le minimum et l'enlève de la liste en un seul passage
- Le prédicat bulle(L1,L2) construit la liste L2, qui contient les mêmes éléments que la liste L1, mais dont le premier élément est inférieur à tous les autres éléments de la liste (le minimum)

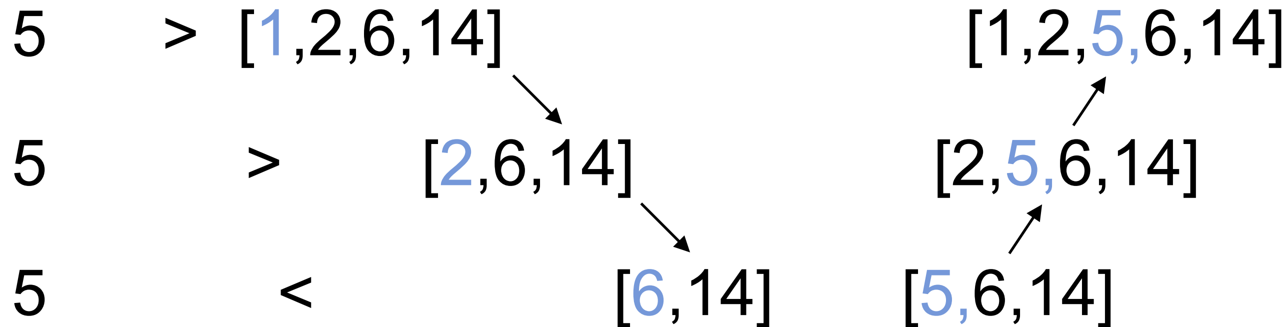
TRI PAR INSERTION : PRINCIPE

- Principe : on trie récursivement la liste privée du premier élément, puis on y insère le premier élément
- Exemple :



INSERTION DANS UNE LISTE TRIÉE

- Principe : on compare l'élément à insérer avec le premier élément de la liste
- Exemple : insérer 5 dans [1,2,6,14]



Prédicat réalisant l'insertion

- insertion(X,L,L1)
  insère l'élément X dans L pour donner L1

```
insertion(X,[],[X]).
insertion(X,[Y|L],[X,Y|L]):- X=<Y.
insertion(X,[Y|L],[Y|L1]):- X>Y,
                             insertion(X,L,L1).
```

Prédicat réalisant le tri

- tri_insertion(L,L1)
  construit la liste triée L1 des éléments de L

```
tri_insertion([],[]).
tri_insertion([X|L],LT):-
                tri_insertion(L,L1),
                insertion(X,L1,LT).
```

Tri par fusion : l'approche « Diviser pour régner »

- Structure récursive : pour résoudre un problème donné, l'algorithme s'appelle lui-même récursivement une ou plusieurs fois sur des sous problèmes très similaires

- Le paradigme « diviser pour régner » donne lieu à trois étapes à chaque niveau de récursivité : diviser, régner, combiner

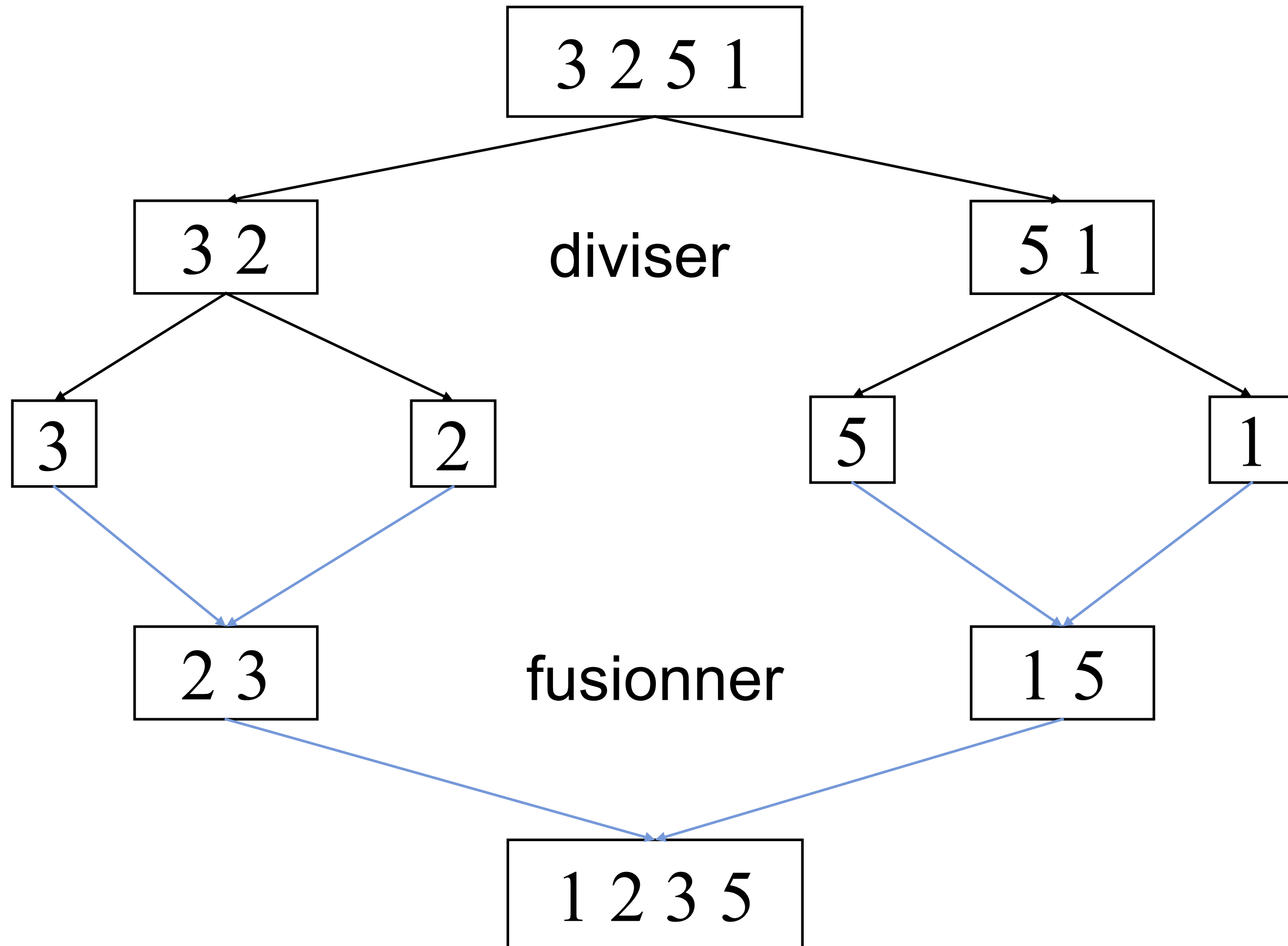
Diviser pour régner : 3 étapes

- Diviser le problème en un certain nombre de sous problèmes
- Régner sur les sous-problèmes en les résolvant récursivement
  Si la taille d'un sous-problème est assez réduite, on peut le résoudre directement
- Combiner les solutions des sous-problèmes en une solution complète pour le problème initial

Tri par fusion : le principe

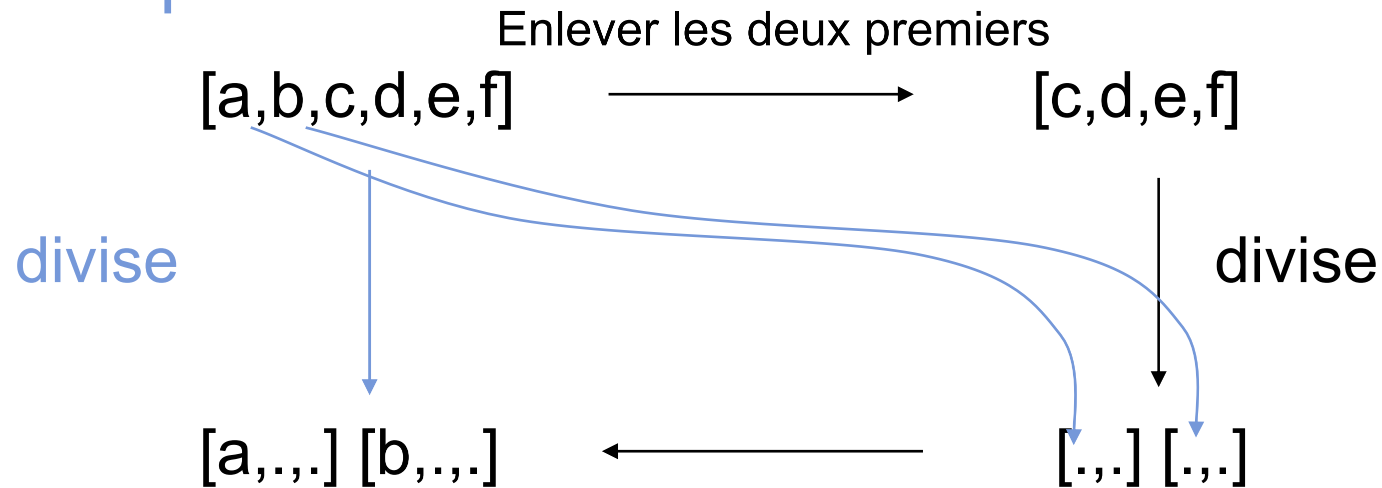
- Diviser : diviser la séquence de n éléments à trier en deux sous-séquences de n/2 éléments
- Régner : trier les deux sous séquences récursivement à l'aide du tri par fusion
- Combiner : fusionner les deux sous-séquences triées pour produire la réponse triée

Un exemple



DIVISER LA LISTE EN DEUX SOUS-LISTES

- Principe :



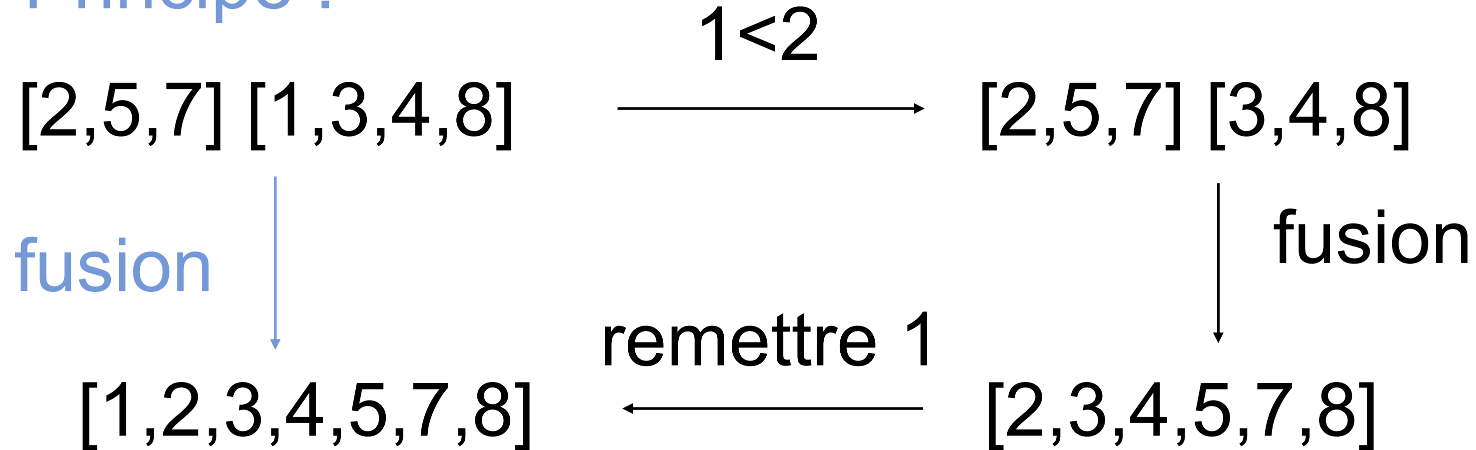
PRÉDICAT DE PARTAGE D'UNE LISTE EN DEUX SOUS-LISTES

- Le prédicat divise(L,L1,L2) divise la liste L en deux sous-listes de taille identique

```
divise([],[],[]).
divise([X],[X],[]).
divise([X,Y|L],[X|L1],[Y|L2]) :- divise(L,L1,L2).
```

Fusionner deux listes triées

- Principe :



Prédicat de fusion

- Le prédicat fusion(L1,L2,L) fusionne les deux listes triées L1 et L2 pour construire la liste triée L

```
fusion([],L,L).
fusion(L,[],L).
fusion([X|L1],[Y|L2],[X|L]) :-      X=<Y,
                                    fusion(L1,[Y|L2],L).
fusion([X|L1],[Y|L2],[Y|L]) :-      X>Y,
                                    fusion([X|L1],L2,L).
```

Prédicat de tri par fusion

- Le prédicat tri_fusion(L,L1) construit la liste triée L1 des éléments de L

```
trifusion([],[]).
trifusion([X],[X]).
trifusion([X,Y|L],T3) :-
      divise([X,Y|L],L1,L2),
      trifusion(L1,T1),
      trifusion(L2,T2),
      fusion(T1,T2,T3).
```

Arbres en Prolog

- Arbres binaires
- Représentation des arbres
- Parcours d’arbres
- Arbres ordonnés

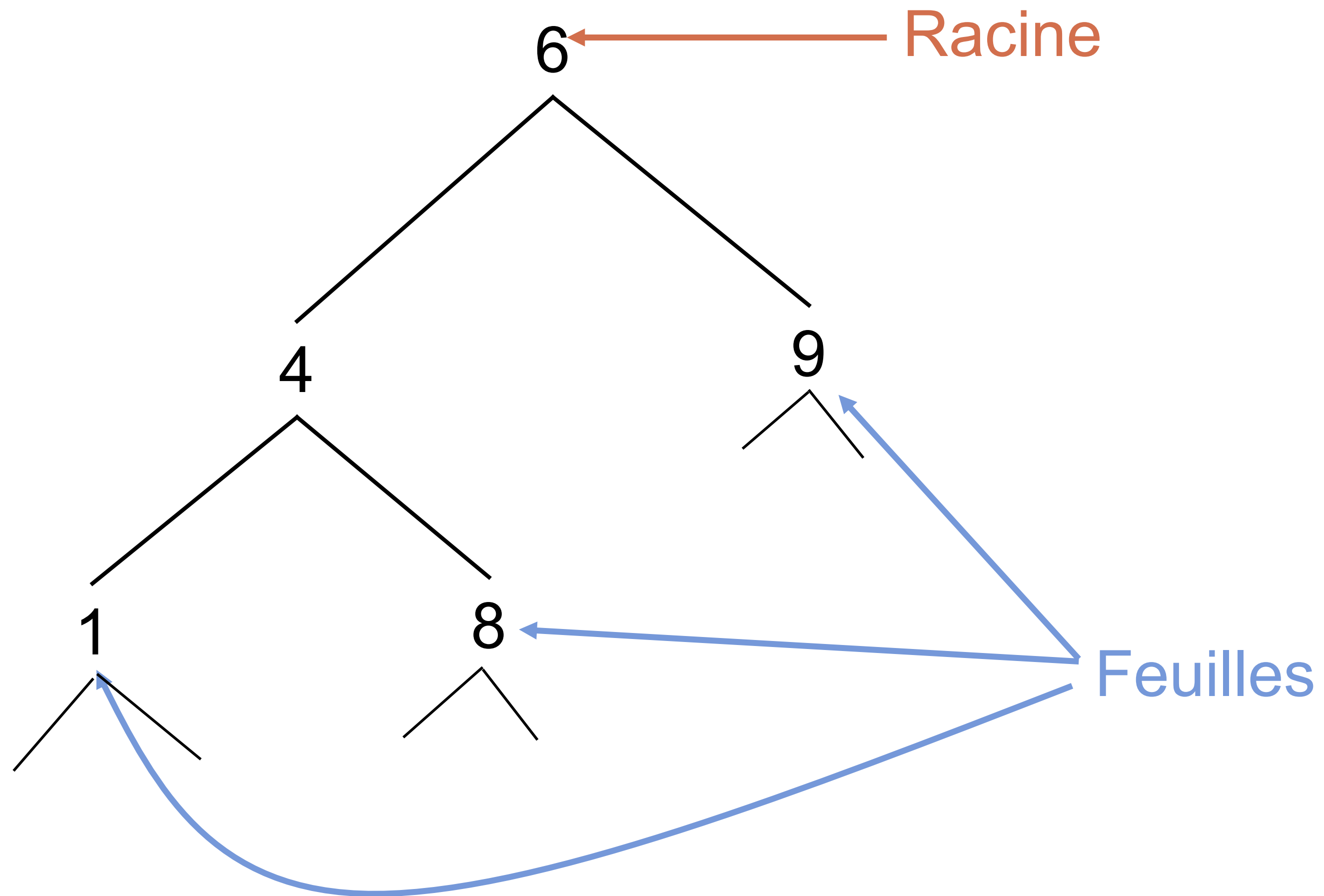
À QUOI SERVENT LES ARBRES ?

- Les arbres, comme les listes, permettent de représenter un nombre variable de données
- Le principal avantage des arbres par rapport aux listes est qu'ils permettent de ranger les données de telle sorte que les recherches soient plus efficaces

Définition

- Un arbre est soit un nœud, soit un arbre vide
- Les nœuds portent des valeurs, ce sont les données que l'on veut stocker
- Un nœud a des fils qui sont eux aussi des arbres
- Si tous les fils d'un nœud sont vides, alors le nœud est qualifié de feuille
- Si tous les nœuds de l'arbre ont n fils, alors l'arbre est dit n-aire

EXEMPLE D'ARBRE



ARBRES BINAIRES

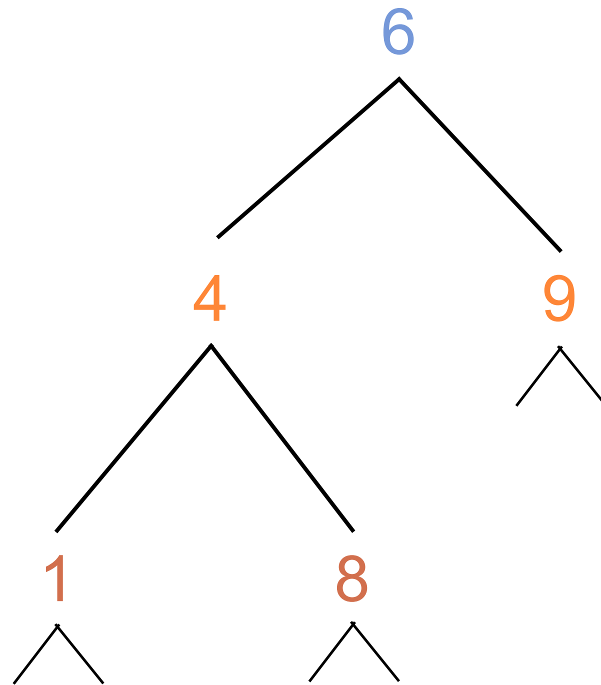
- Un arbre binaire est :
  - soit l'arbre vide
  - soit un nœud qui a exactement deux fils (éventuellement vides)

Représentation des arbres binaires

- Nous choisissons d'utiliser les listes pour représenter les arbres
- Un arbre vide sera représenté par la liste vide []
- Un nœud sera une liste de 3 éléments
  - le premier est sa valeur
  - le deuxième son fils gauche
  - le troisième son fils droit

Exemple de représentation d'un arbre binaire

[6,[4,[1,[],[]],[8,[],[]]],[9,[],[]]]



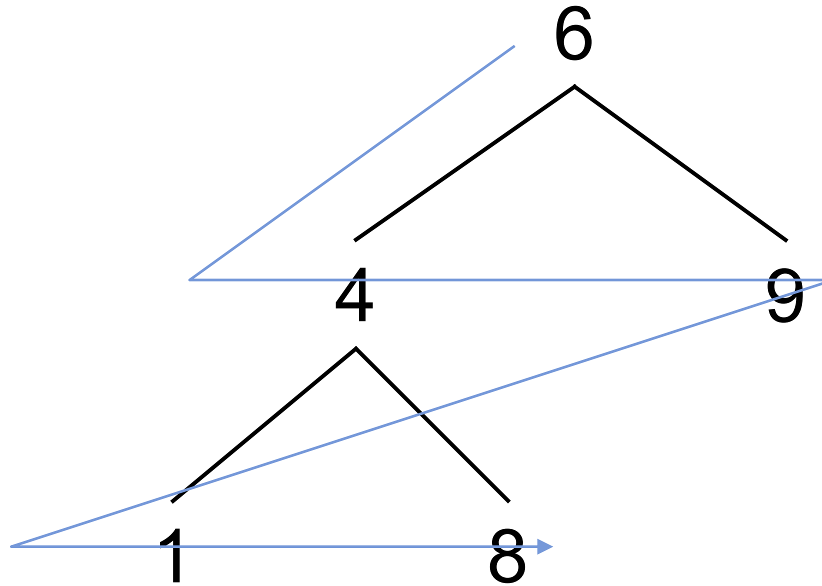
Exercice

- Définir un prédicat qui vérifie qu'une liste représente bien un arbre binaire.

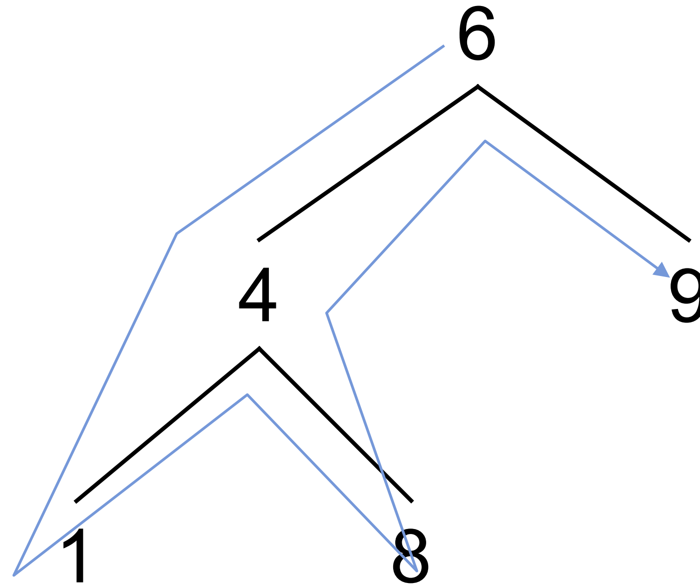
Parcours d'arbres

- Un arbre contient un ensemble de données
- Pour utiliser ces données, il faut parcourir l'arbre : en profondeur ou en largeur

Parcours en largeur



Parcours en profondeur : un exemple



Parcours en profondeur : principe

- Parcourir un arbre en profondeur consiste à passer ses nœuds en revue, en commençant toujours par le même fils, et en descendant le plus profondément possible dans l'arbre
- Lorsque l'on arrive sur un arbre vide, on remonte jusqu'au nœud supérieur et on redescend dans le fils encore inexploré

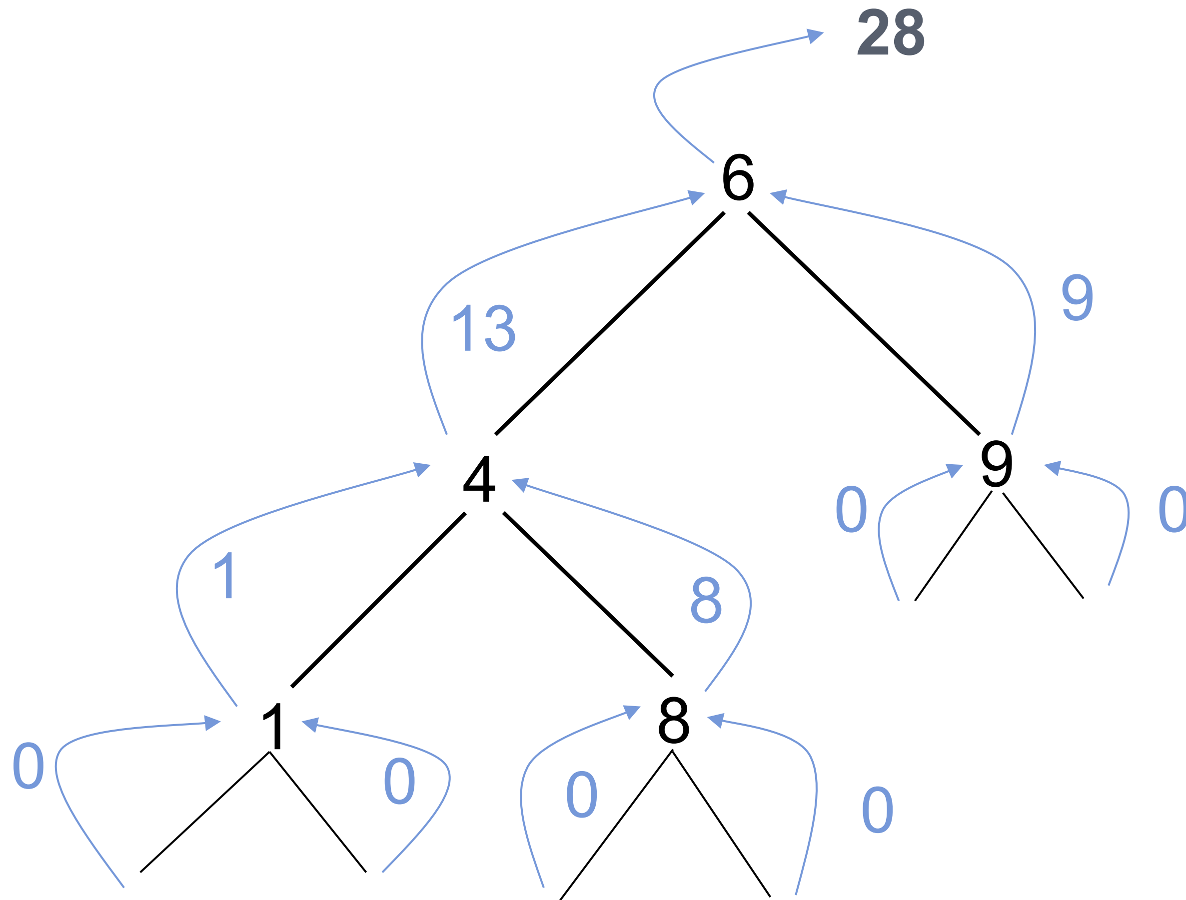
Pour écrire un algorithme qui effectue un parcours en profondeur

- Pour écrire un prédicat P, sur un arbre A
  - Si A est vide, on retourne une valeur constante
  - Si A n'est pas vide :
    - on rappelle P sur les deux fils de A, ce qui retourne deux résultats : Rg et Rd
    - puis on retourne un résultat qui ne dépend que de Rg, Rd et de la valeur de A

Exemple : somme des valeurs d'un arbre de nombres

```
somme([],0).
somme([N,G,D],M) :-
    somme(G,N1),
    somme(D,N2),
    M is N+N1+N2.
```

FONCTIONNEMENT SUR UN EXEMPLE



Exercices

- Définir un prédicat qui calcule la hauteur d'un arbre binaire :
  - La hauteur d'une feuille est zéro
  - La hauteur d'un nœud est 1+max(hauteur_fils_gauche, hauteur_fils_droit)
- Définir un prédicat qui calcule le maximum d'un arbre binaire de nombres.
- Définir un prédicat qui calcule la liste résultant du parcours infixe d'un arbre binaire.

Modification et création d'arbres

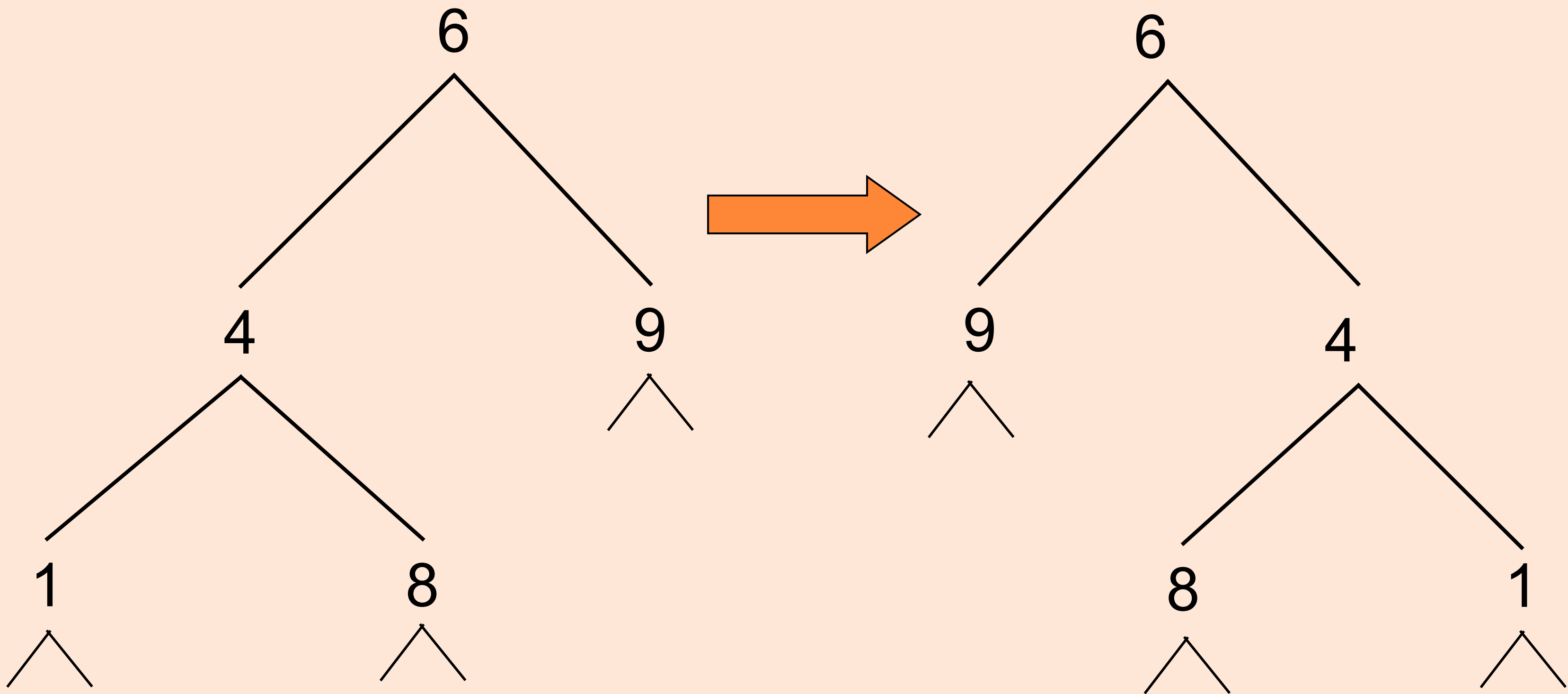
- Exemple : écrire un prédicat qui ajoute 1 à tous les nœuds d'un arbre qui contient des nombres
- Il ne s'agit pas d'une modification (ajouter 1), mais d'une création : écrire un prédicat qui construit un arbre identique à celui donné, mais dans lequel on a ajouté 1 à tous les nœuds

PRÉDICAT AJOUTE1

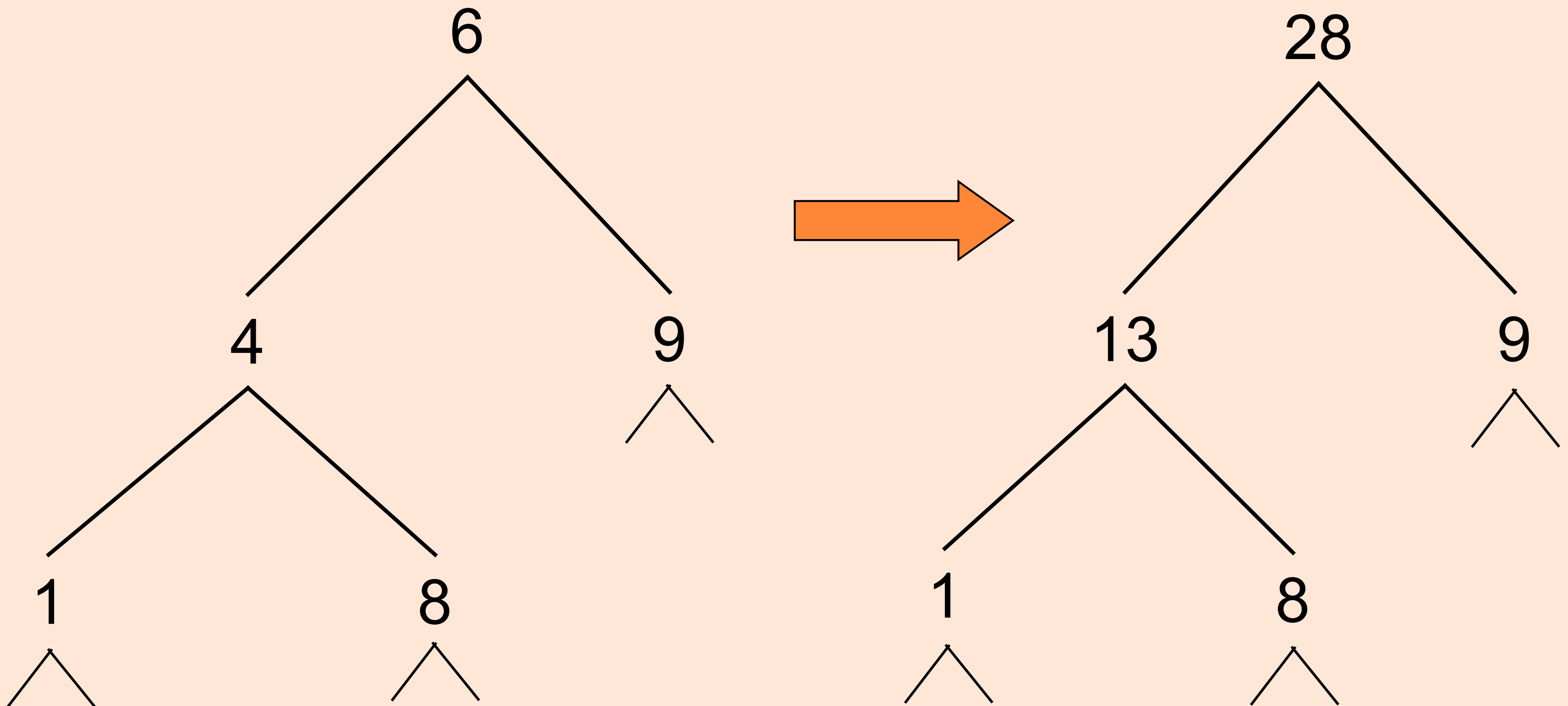
```
ajoute1([],[]).
ajoute1([N,G,D],[N1,G1,D1]) :-
        N1 is N+1,
        ajoute1(G,G1),
        ajoute1(D,D1).
```

Exercice

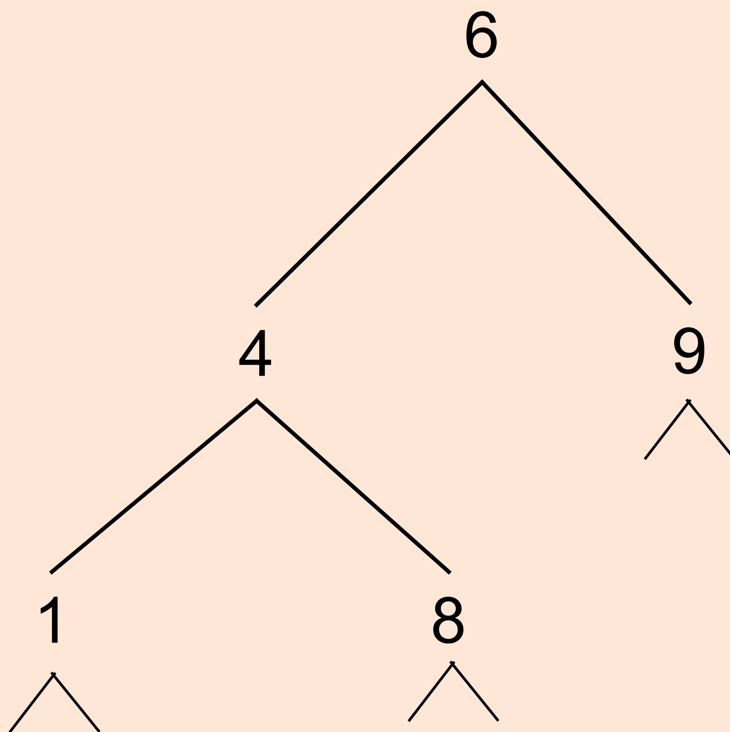
- Définir un prédicat qui construit le miroir d'un arbre binaire

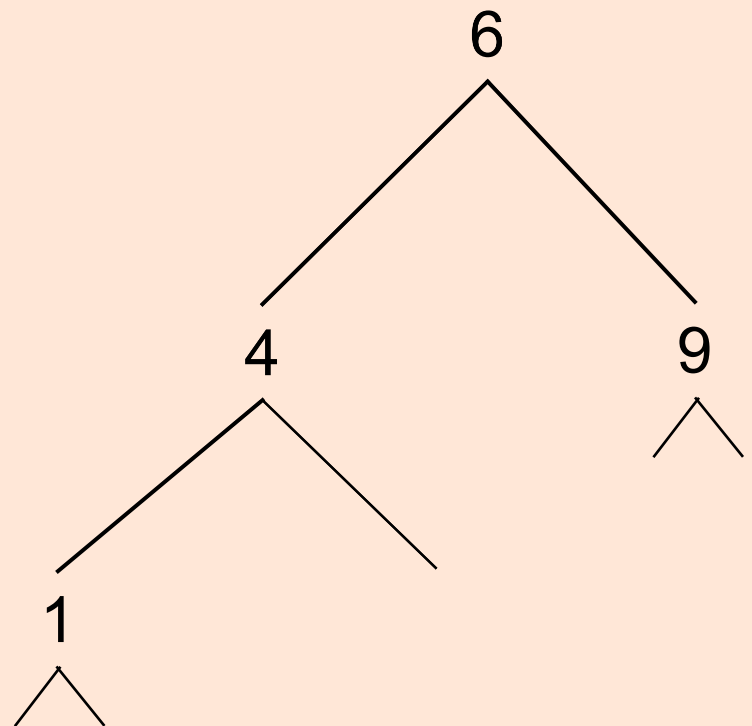


Somme des valeurs des fils



TESTS

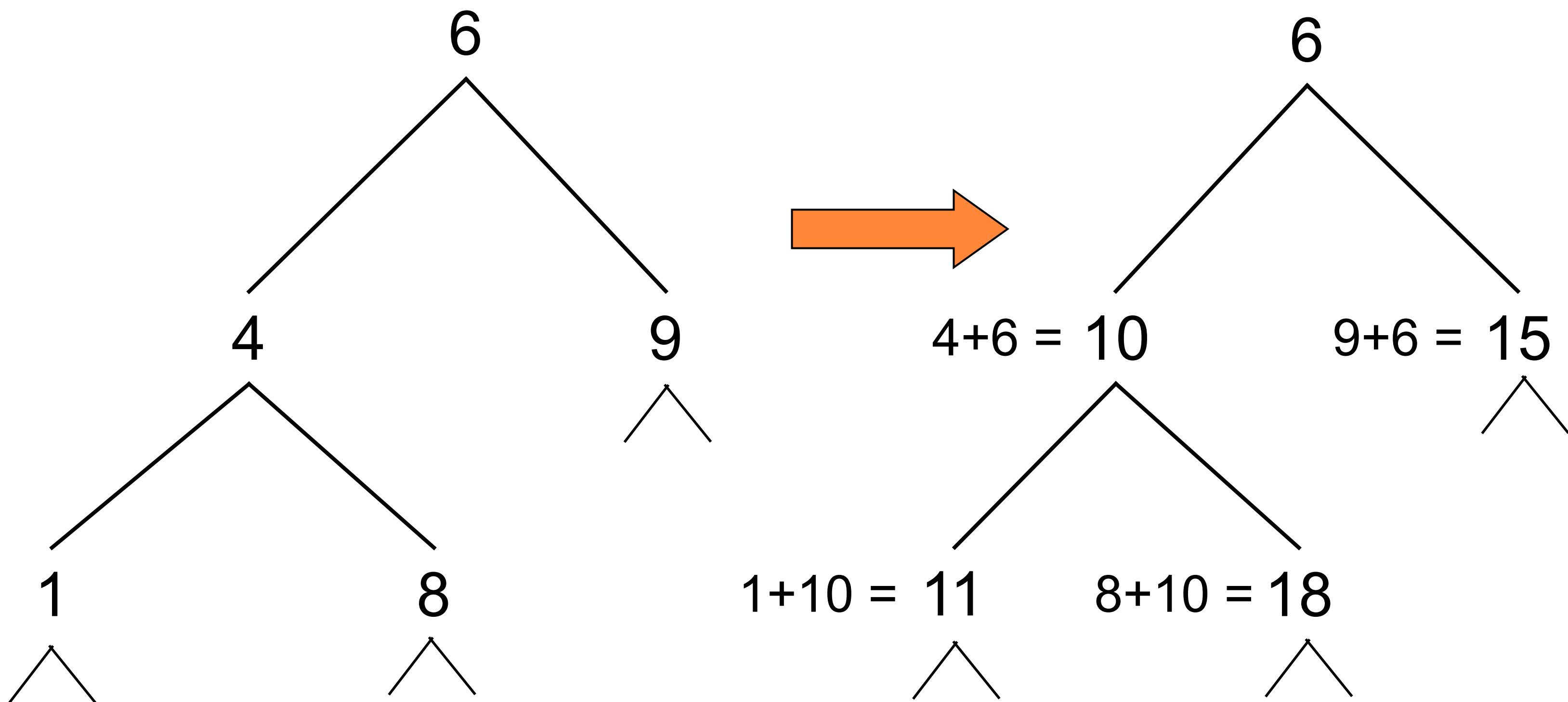




PRÉDICAT SOMME-FILS

```
somme_fils([],[]).
somme_fils([N,[],[]],[N,[],[]]).
somme_fils([N,G,[]],[N1,[G1,GG,GD],[]]) :- G\==[],
   somme_fils(G,[G1,GG,GD]), N1 is N+G1.
somme_fils([N,[],D],[N1,[],[D1,DG,DD]]) :- D\==[],
   somme_fils(D,[D1,DG,DD]), N1 is N+D1.
somme_fils([N,G,D],[N1,[G1,GG,GD],[D1,DG,DD]]) :-
   somme_fils(G,[G1,GG,GD]), somme_fils(D,[D1,DG,DD]),
   N1 is N+G1+D1.
```

SOMME DES VALEURS DES PÈRES



Calcul en remontant ou en descendant

- Dans tous les prédicats précédemment écrits, le résultat dépendait des fils
  $\Rightarrow$ calcul en remontant
- Ici, le résultat dépend du père
  $\Rightarrow$ calcul en descendant
  $\Rightarrow$ paramètre supplémentaire pour passer le résultat du père au fils

PRÉDICAT SOMME_PERE

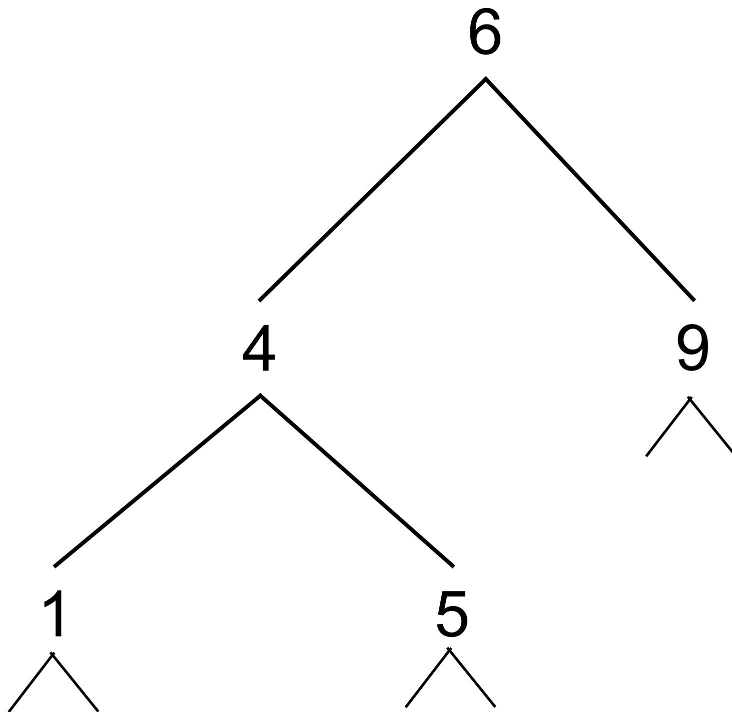
```
somme_pere(A,A1) :-
   somme_pere2(A,0,A1).

somme_pere2([],_,[]).
somme_pere2([N,G,D],V,[N1,G1,D1]) :-
   N1 is N+V,
   somme_pere2(G,N1,G1),
   somme_pere2(D,N1,D1).
```

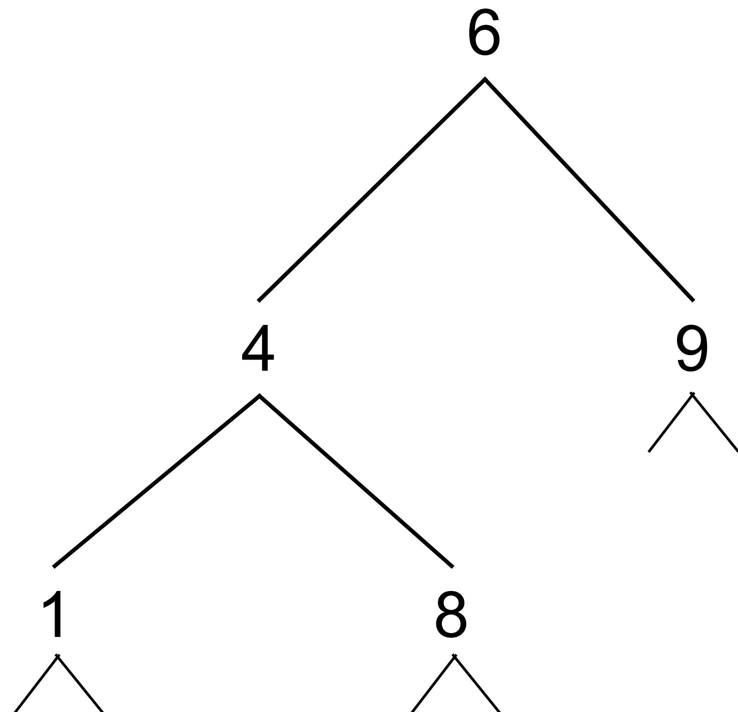
ARBRES BINAIRES DE RECHERCHE (OU ORDONNÉS)

- Les valeurs des nœuds peuvent être ordonnées
- En chaque nœud de l'arbre, la valeur du nœud est :
  - supérieure à toutes celles de son fils gauche
  - inférieure à toutes celles de son fils droit

Exemples



Arbre ordonné

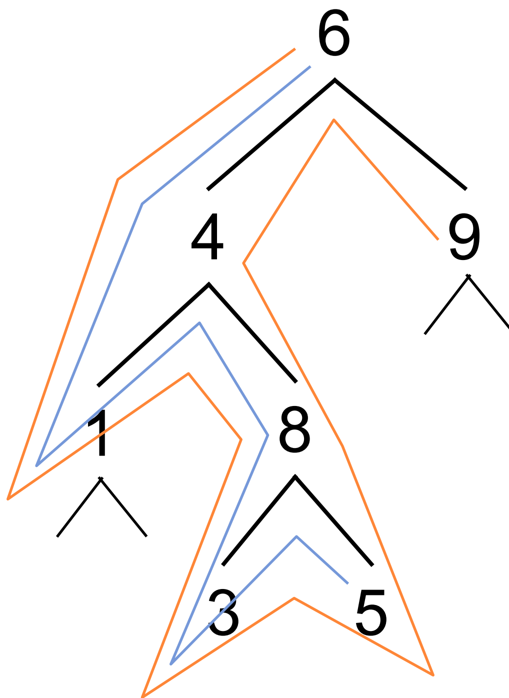


Arbre non ordonné

Recherche d'un élément dans un arbre binaire quelconque

- Principe : tant qu'on n'a pas trouvé la valeur V, il faut comparer V avec toutes les valeurs de l'arbre A

EXEMPLE



Recherche fructueuse :
Chercher 5

Cas le pire
Recherche infructueuse :
Chercher 7

Complexité au pire :
nombre de nœuds de l'arbre

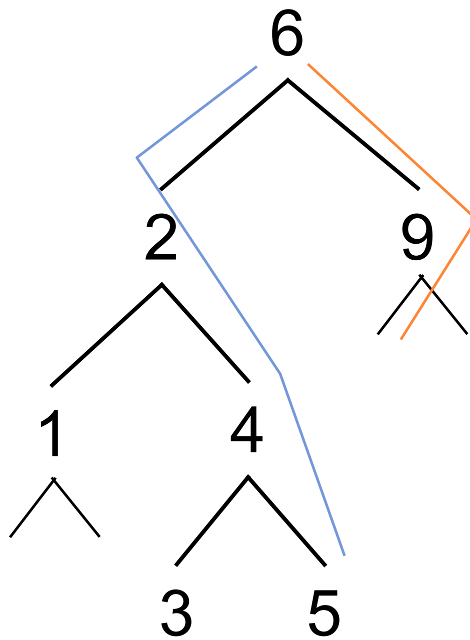
Recherche d'un élément dans un arbre binaire de recherche (1)

- Principe : utiliser le fait que l'arbre est ordonné pour choisir dans quelle branche de l'arbre chercher

Recherche d'un élément dans un arbre binaire de recherche (2)

- Algorithme :
  - Cas d'arrêt :
    - Si A est vide Alors Retourne Faux
    - Si valeur(A)=V Alors Retourne Vrai
  - Appels récursifs :
    - Si V>valeur(A) Alors chercher V dans fils-droit(A)
    - Si V<valeur(A) Alors chercher V dans fils-gauche(A)

EXEMPLE



Recherche fructueuse :
Chercher 5

Recherche infructueuse :
Chercher 7

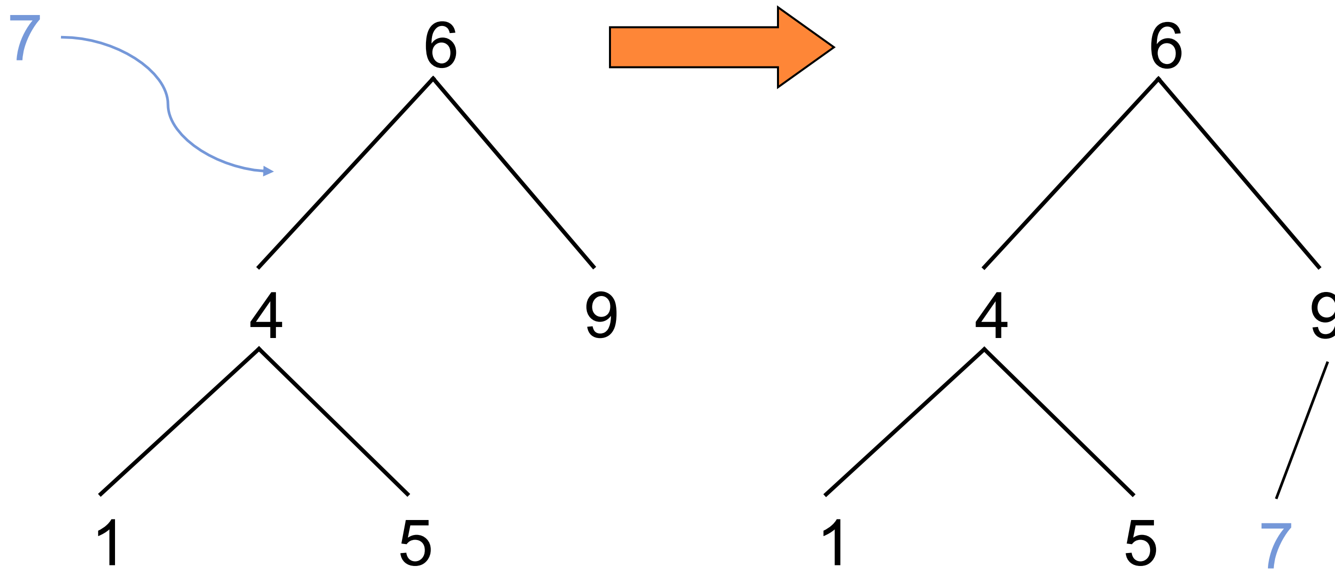
Complexité au pire :
hauteur de l'arbre

Exercices

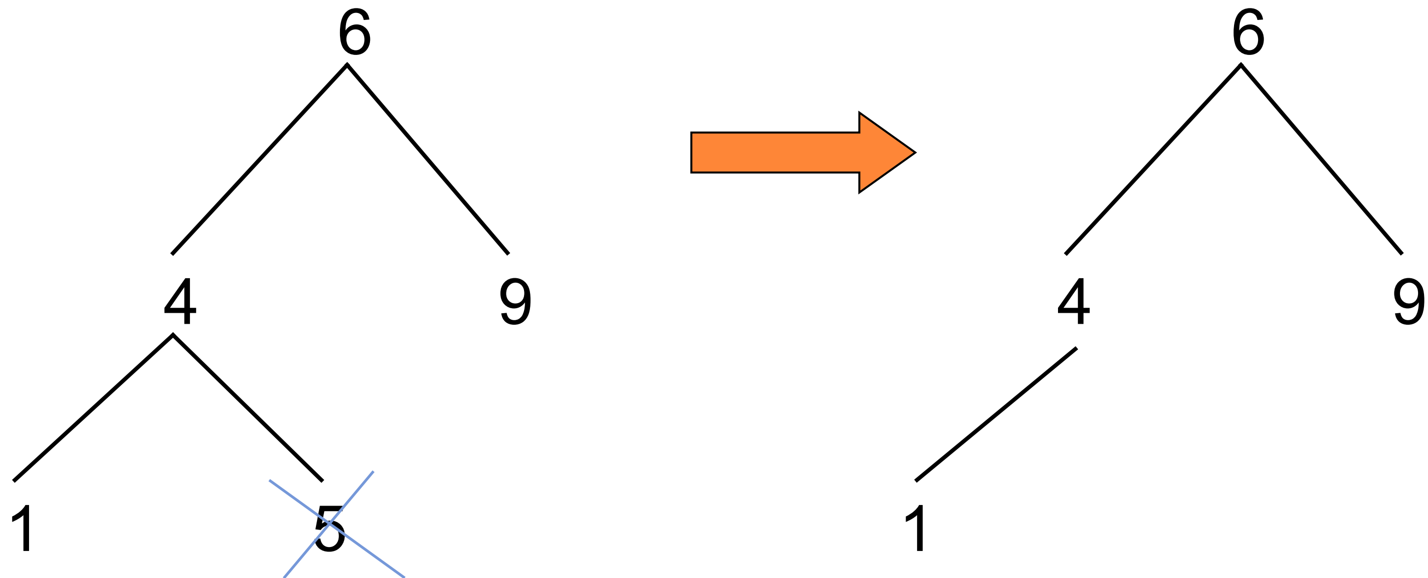
- Définir le prédicat de recherche d'un élément dans un arbre binaire de recherche
- Définir un prédicat qui calcule le maximum d'un arbre binaire de recherche

INSERTION DANS UN ABR

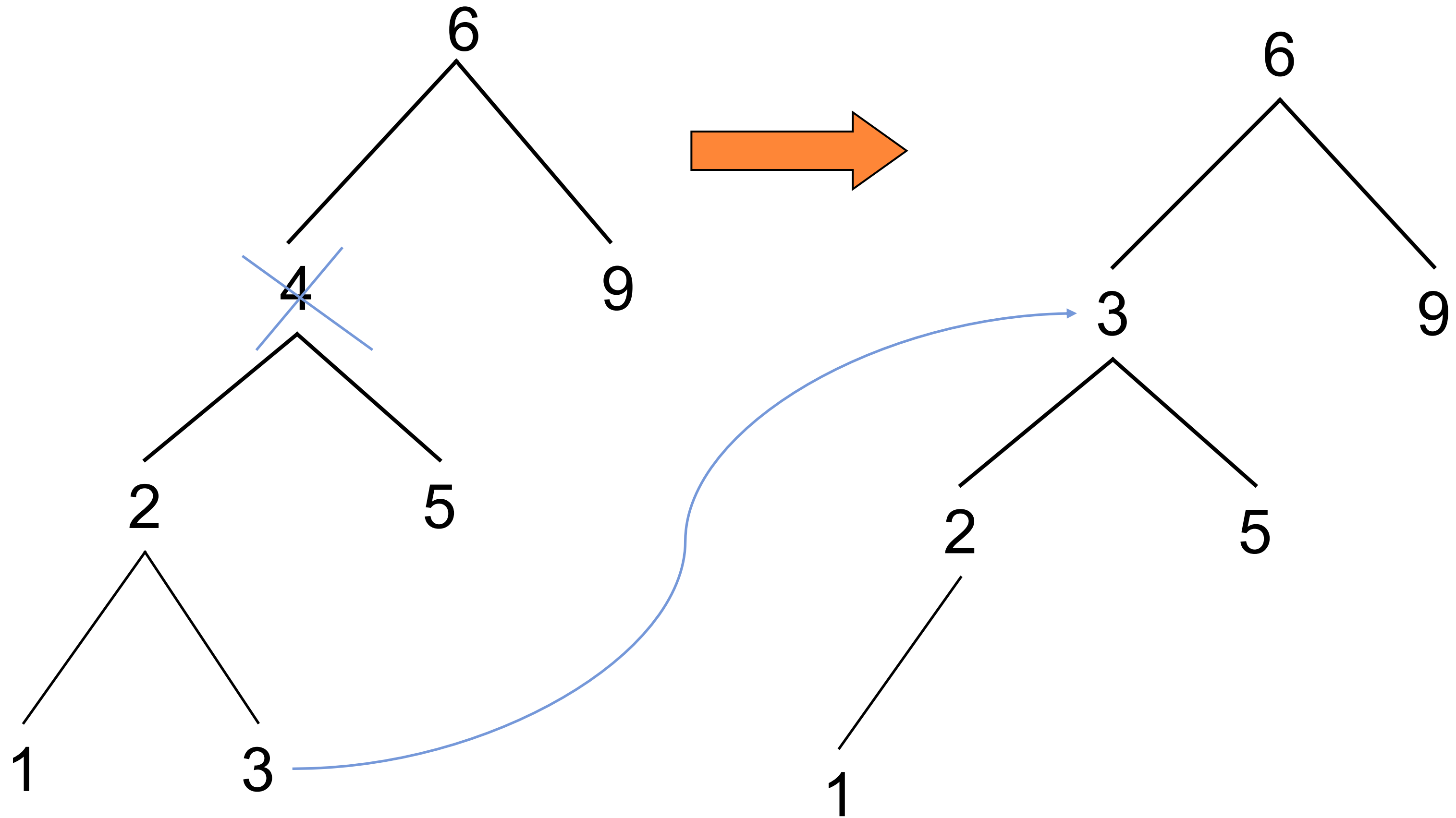
- Principe : on insère en bout de branche



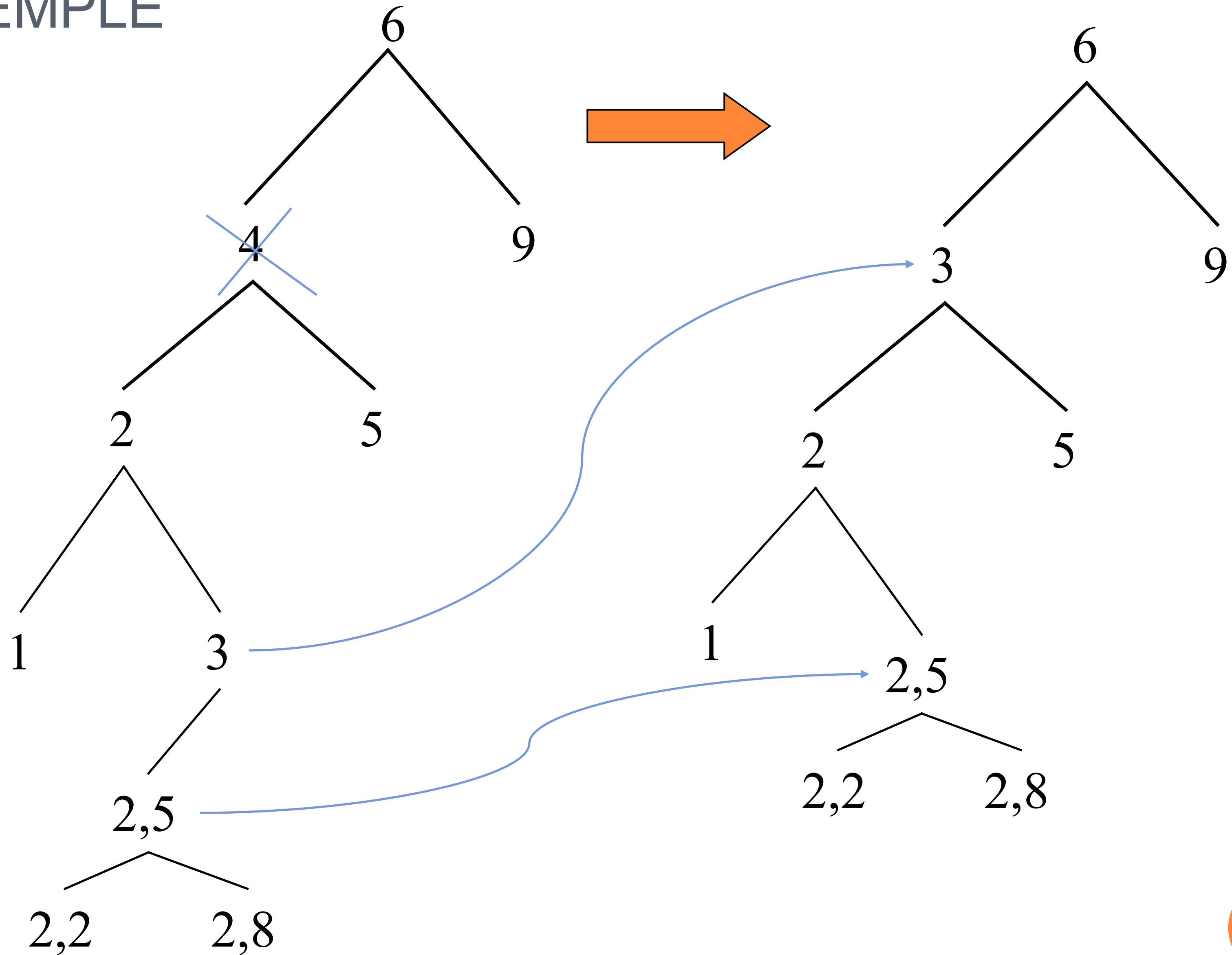
SUPPRESSION DANS UN ABR : EXEMPLE



Suppression dans un ABR : exemple



Exemple



Suppression dans un ABR

- Pour supprimer la valeur V dans un ABR
  - Si V est une feuille, alors on supprime la feuille
  - Sinon on remplace la valeur V par la valeur V' qui lui est immédiatement inférieure (ou immédiatement supérieure), de manière à respecter l'ordre, puis on supprime V' qui est le plus grand élément du fils gauche de V (resp. le plus petit élément de son fils droit)
  - V' est une feuille ou un élément qui n'a pas de fils droit (resp. pas de fils gauche), et peut donc être supprimée facilement