IFT339
Structures de données

Thème 11 : Arbres balancés, Arbres AA
Arbres rouges et noirs

Aïda Ouangraoua
Département d'informatique

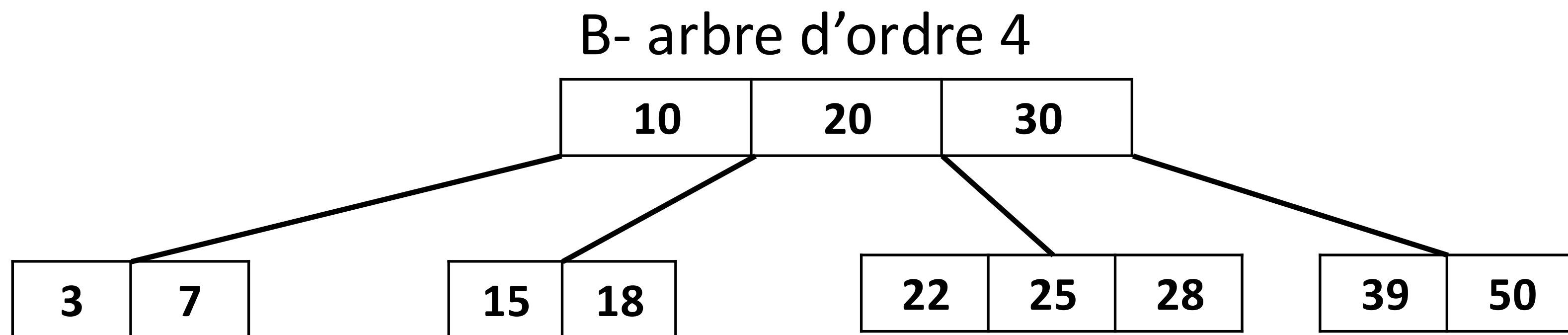
UNIVERSITÉ DE
SHERBROOKE

Arbres balancés (B-arbre)

- Avec AVL : recherche en O(log(n))
- Comment faire mieux pour des ensembles très grands (n très grand)
- Solution : relâcher la contrainte d'arbre binaire pour des avoir des arbres plus larges et de faible hauteur
- Contrainte: toutes les feuilles doivent avoir la même hauteur

Arbres balancés (B-arbre) d'ordre m

- Chaque nœud a entre m/2 et m enfants (la racine, entre 2 et m enfants)
  - Un nœud qui a k enfants contient k-1 éléments
  - Entre chaque paire d'éléments consécutifs, il y a un enfant (sous-arbre)



- Nombre min/max d'éléments dans un B-arbre d'ordre 100 et de hauteur 3

  max = 99 + 100*99 + 100*100*99

  min = 1 + 2*49 + 2*50*49
- Pour un AVL de hauteur 3 : max = 7 ; min = 3

Insertion de x dans un B-arbre

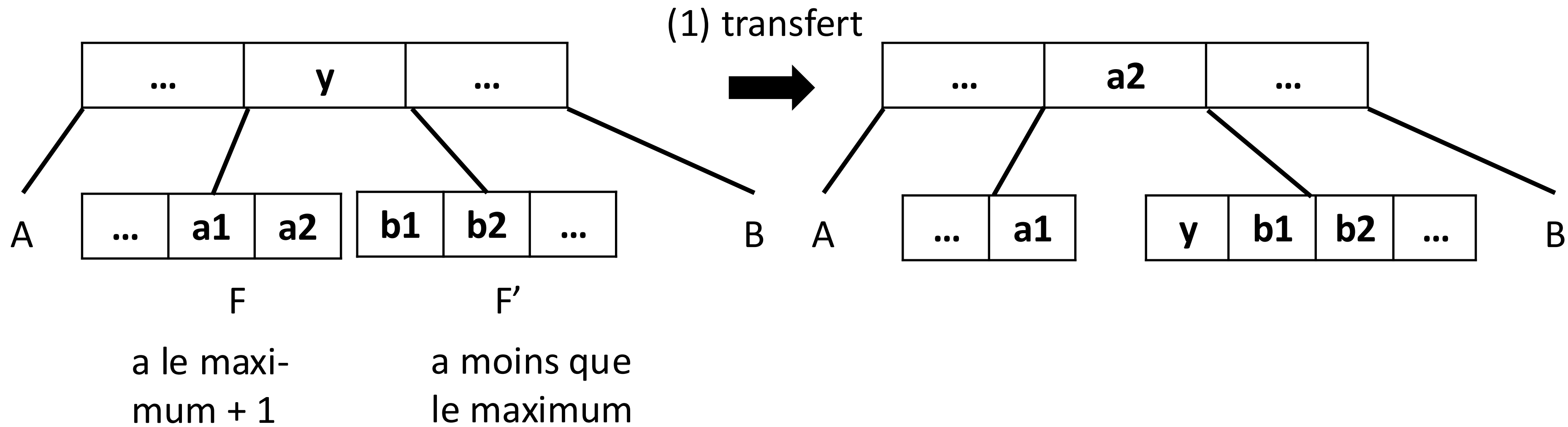
- Localiser la feuille F où insérer
- Insérer x dans F
- Tant que F est saturé, i.e. F a plus de m-1 éléments:

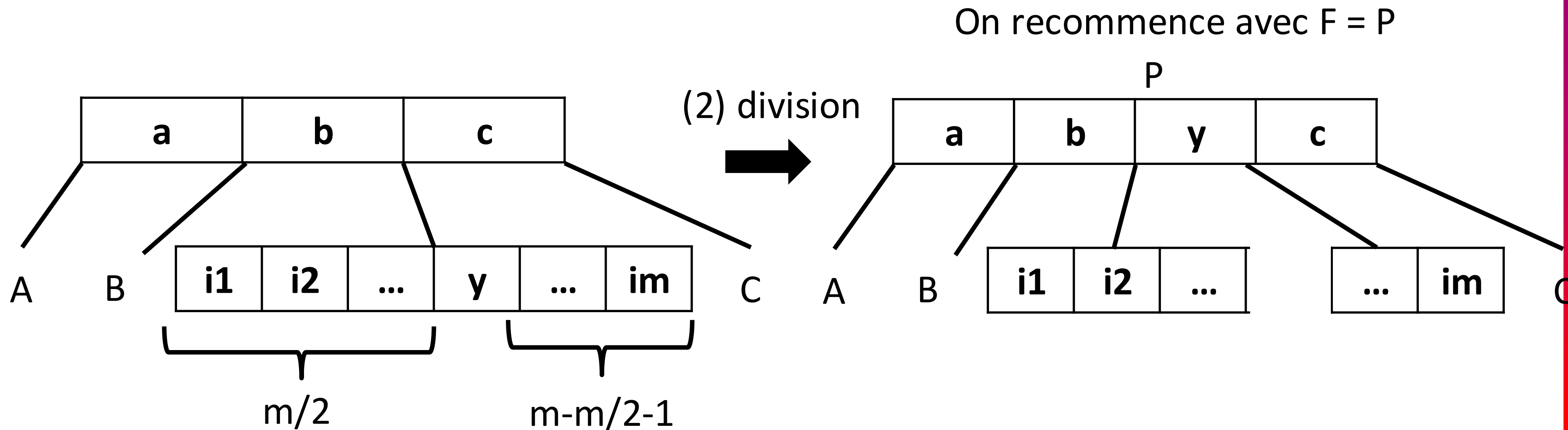
On insère toujours dans une feuille

Facultatif

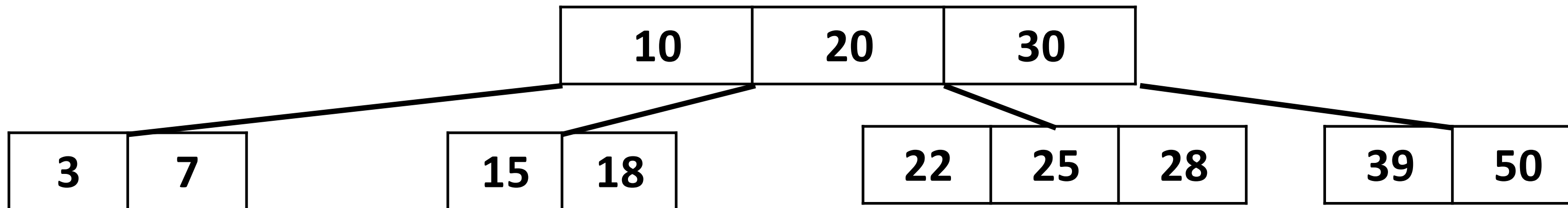
  - (1-Transfert) Si F a un frère adjacent F' qui a moins que m -1 éléments
    - Transférer un élément de F vers le parent et un élément du parent vers F'
  - (2-Division) Sinon:
    - Diviser F en deux feuilles F1 et F2 telles que F1 contient les m/2 premiers éléments F2 les m- m/2 – 1 derniers et y est l'élément du milieu
    - Insérer y dans le parent P de F avec F1 comme enfant de P à gauche de y, et F2 comme enfant de P à droite de y
    - F = P

Insertion de x dans un B-arbre

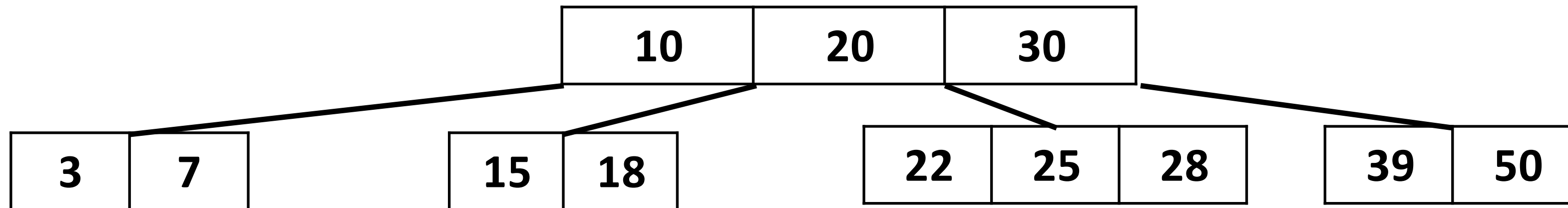


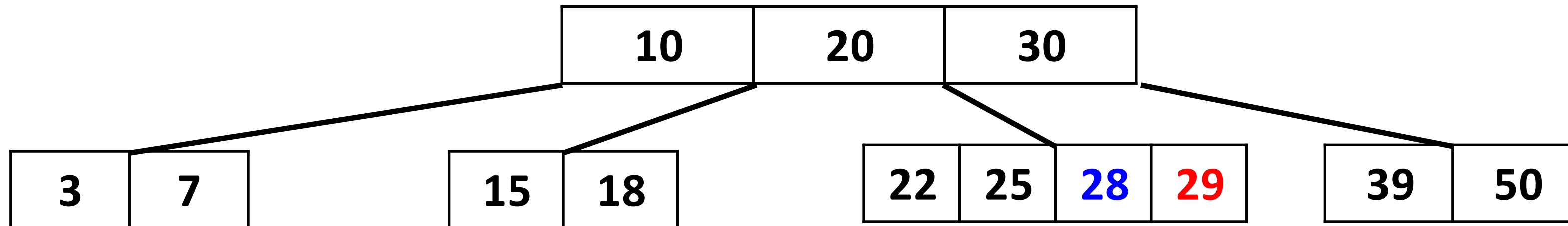


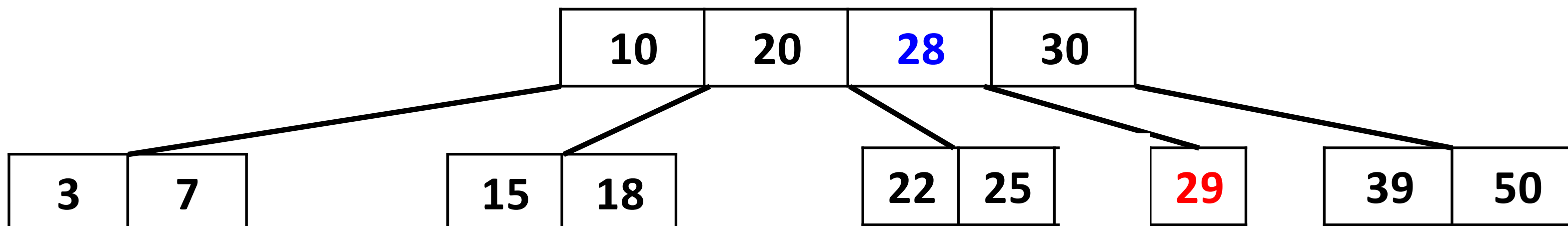
Exemple: Insertion de 29 (ordre 4)

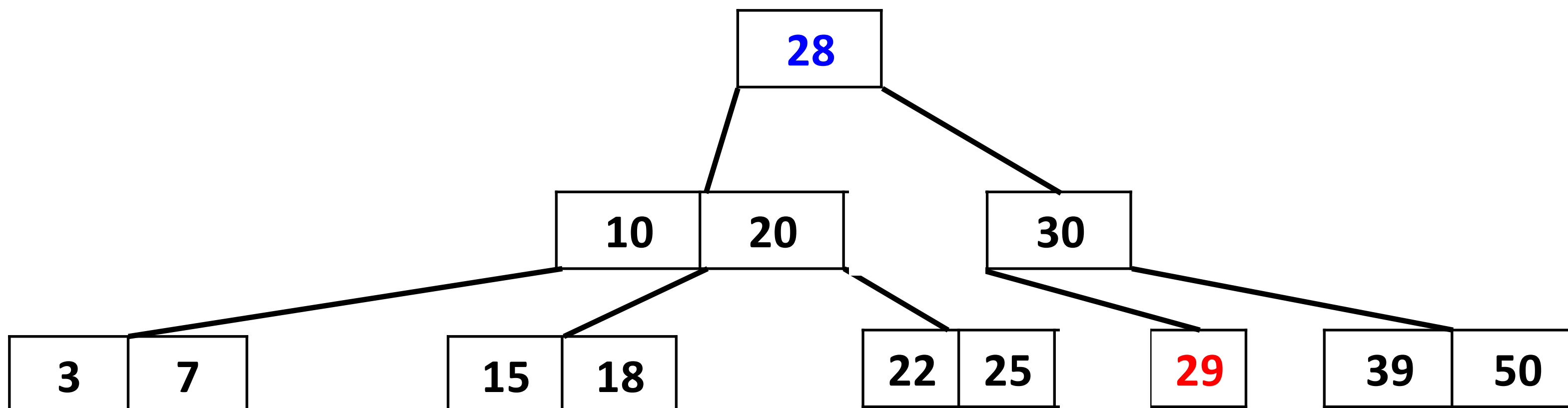


Exemple: Insertion de 29 (ordre 4)









Autre algorithme d’insertion pour m > 2: la taupe

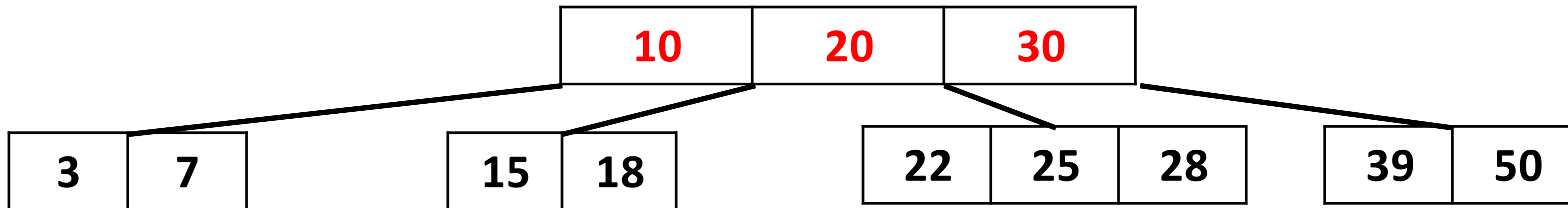
- Ne jamais descendre dans un nœud plein:

- À partir de la racine, descendre vers la feuille F où insérer x
- Pour chaque nœud N rencontré:
  - Si N est plein, i.e N a m-1 éléments:
    - Diviser N en en deux feuilles N1 et N2 telles que
      N1 contient les (m-1)/2 premiers éléments
      N2 les (m-1)- (m-1)/2 – 1 derniers
      et y est l’élément du milieu
    - Insérer y dans le parent de N
- Insérer x dans le dernier nœud (feuille) F

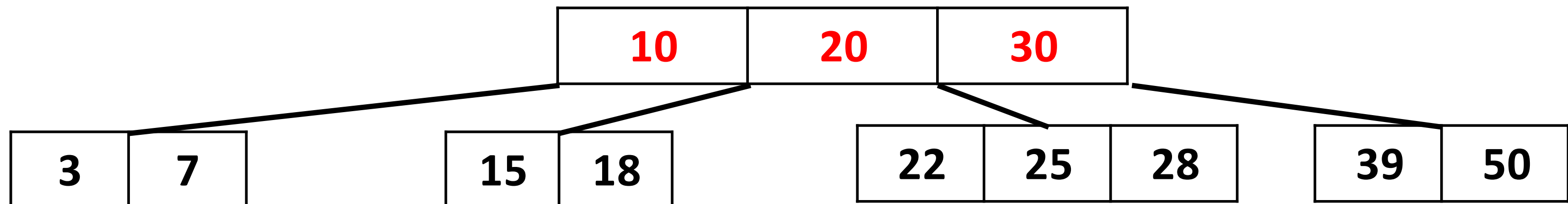
Aucun nœud sur le chemin de la racine à F ne peut être saturé, puisque tous les nœuds plein sur le chemin on été divisés.

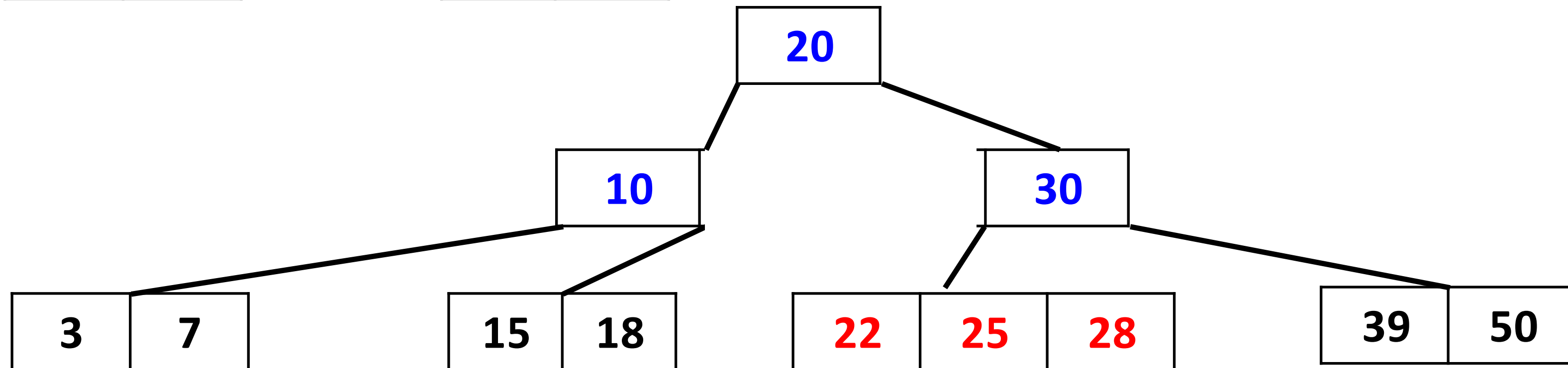
On insère toujours dans une feuille

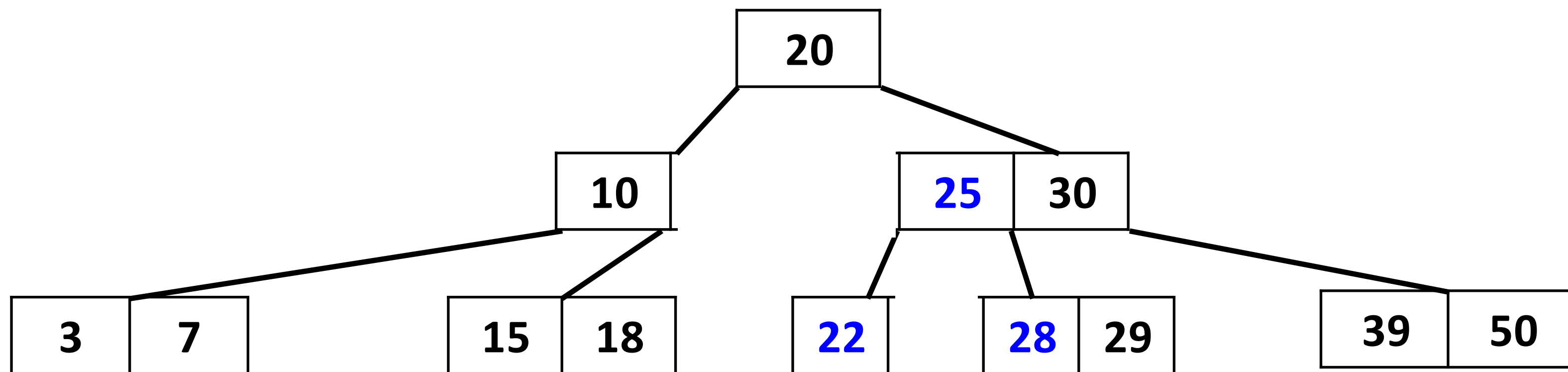
Exemple: Insertion de 29 (ordre 4)



Exemple: Insertion de 29 (ordre 4)





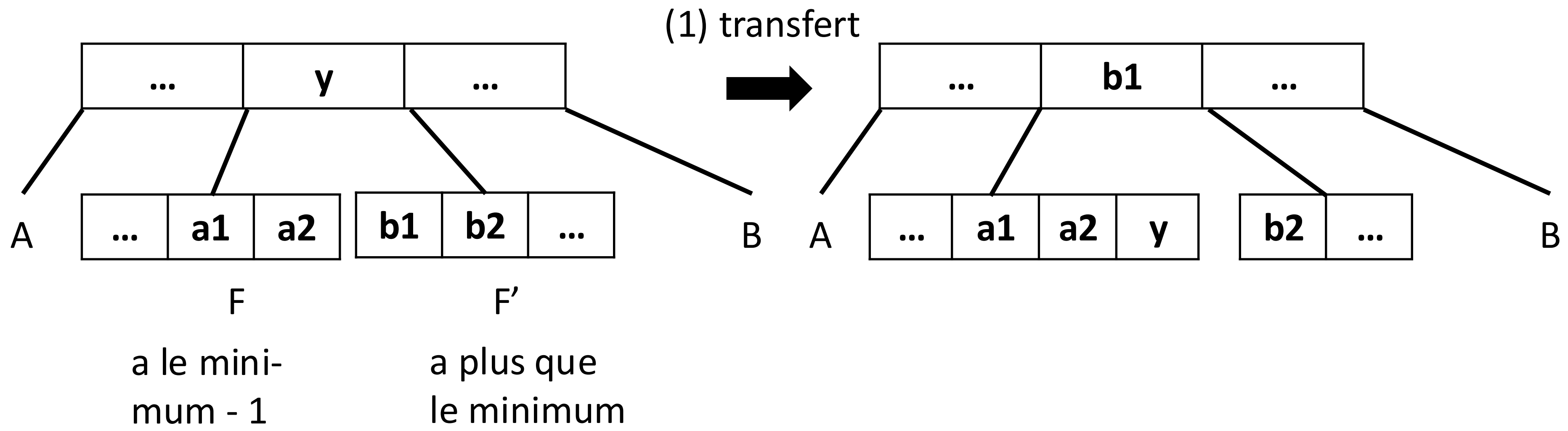


Suppresion de x dans un B-arbre

- Localiser le nœud N où se trouve x
- Si N est une feuille
  - F = N; Supprimer x de F
- Sinon
  - localiser la feuille F contenant le précédent x' de x
  - Échanger les valeurs de x et x'
  - Supprimer x de F
- Tant que F est sous-rempli, i.e. F a moins de m/2 - 1 éléments:
  - (1-Transfert) Si F a un frère adjacent F' qui a plus de m/2 -1 éléments
    - Transférer un élément de F' vers le parent et un élément du parent vers F
  - (2-Fusion) Sinon:
    - Fusionner de F avec un un frère adjacent F' + un élément du parent P
    - F = P

On supprime toujours dans une feuille

Suppresion de x dans un B-arbre



On recommence avec F = P

P

(2) fusion

...
y
...

A ... a1 a2 b1 b2 ... B

F

a le mini-
mum - 1

F’

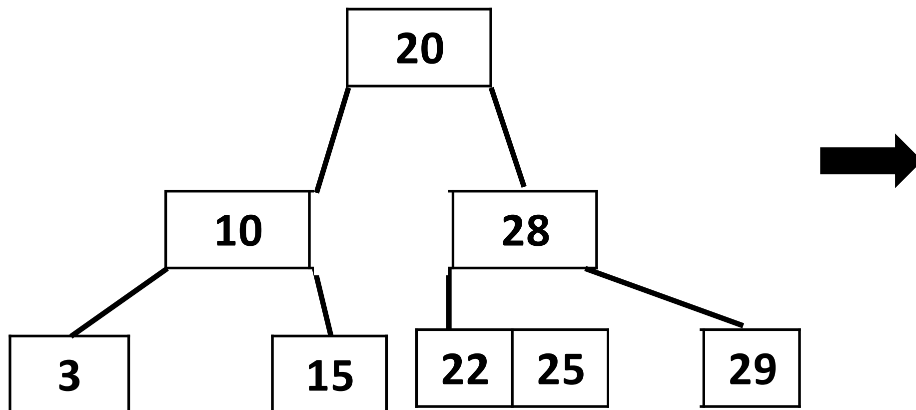
a exactement
le minimum

...
...

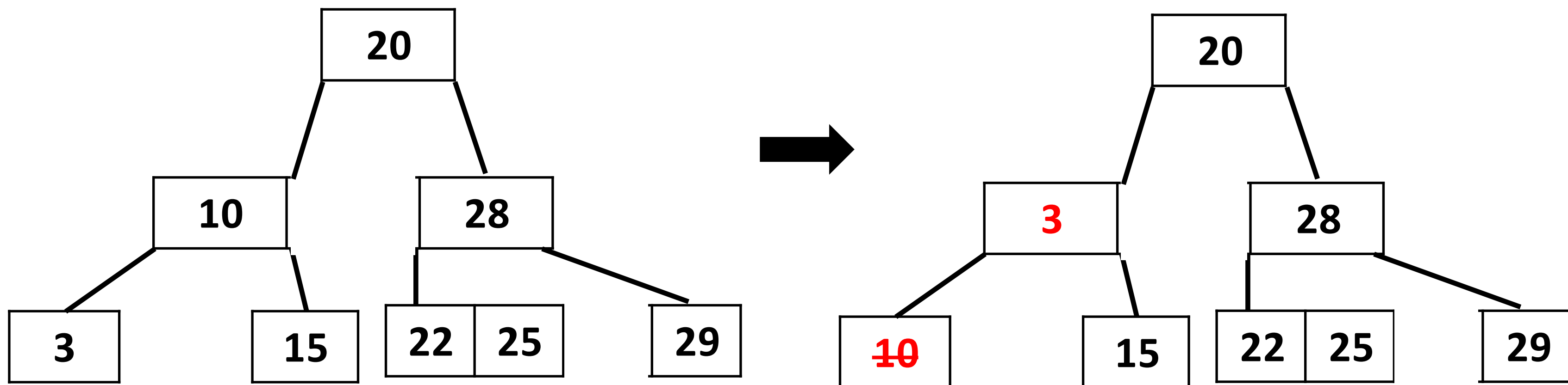
A ... a1 a2 y b1 b2 ... B

F+F’+y

Exemple: Suppresion de 10 (ordre 3)



Exemple: Suppresion de 10 (ordre 3)



Cas(2)

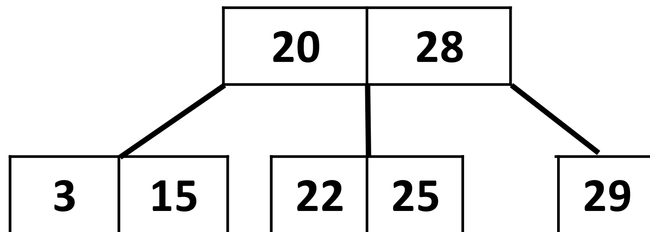
20 28
3 15
22 25
29

-
20 28
3 15
22 25
29

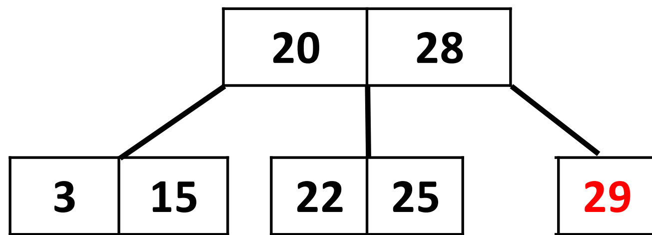
Cas(2)

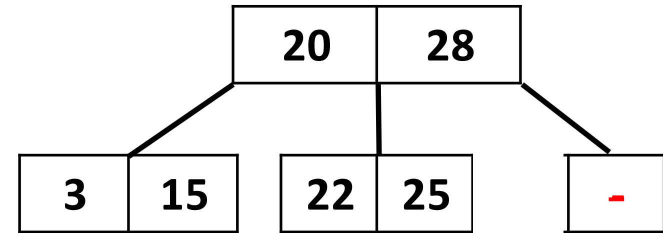
20
-
28
3 15
22 25
29

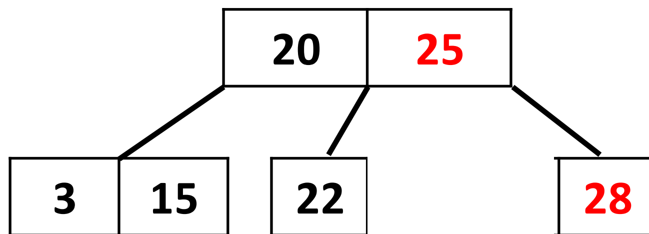
Exemple: Suppresion de 29 (ordre 3)

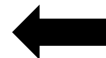


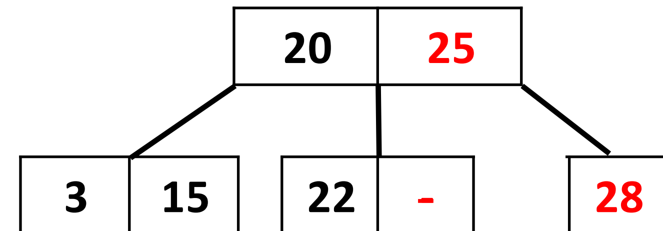
Exemple: Suppresion de 29 (ordre 3)











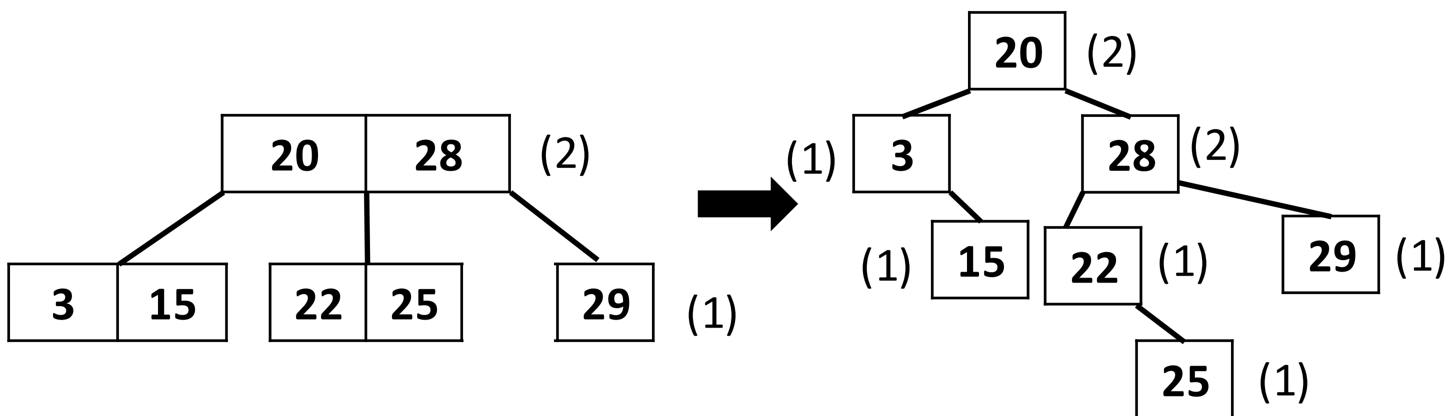
Arbres AA

- B-arbre donne une meilleure complexité pour la recherche que les arbres AVL, mais la taille variable des nœuds (entre m/2 -1 et m-1) est difficile à manipuler
- Solution tout B-arbre peut être représenter en arbre binaire
  - Scinder chaque nœud en autant de niveaux qu'il a d'éléments
  - Associer à chaque élément son niveau initial (1 pour une feuille)

n1 ... nk (i)
A1 A2 Ak+1

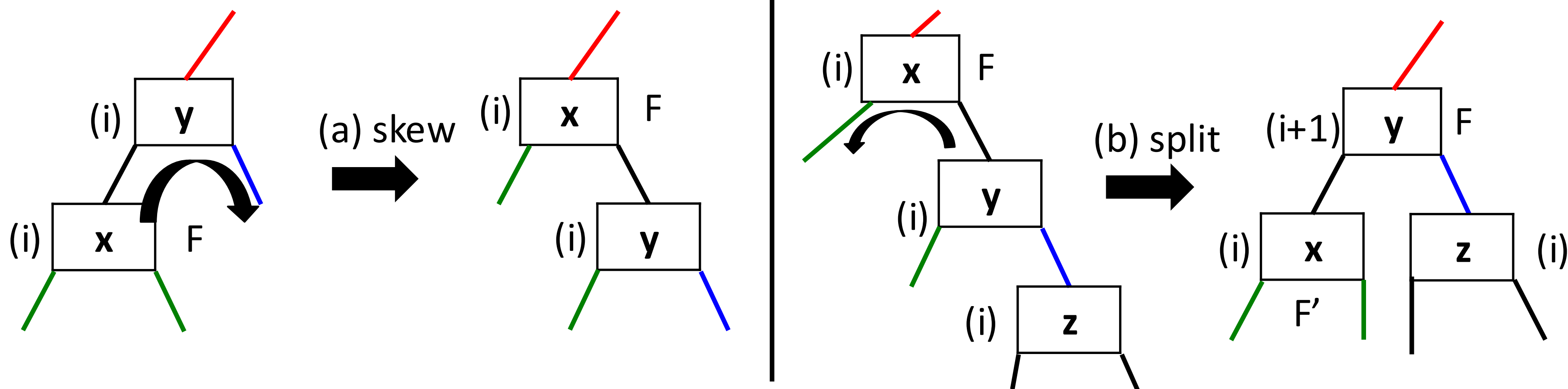
n1 (i)
A1
n2 (i)
A2
nk (i)
Ak Ak+1

- Arbre AA : Arbre binaire correspondant à un **B-arbre d'ordre 3**.

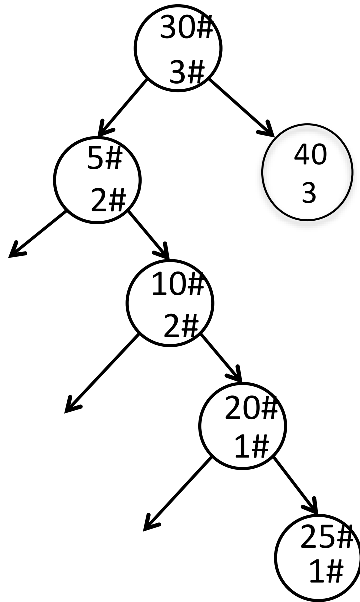


Insertion de x dans un AA-arbre

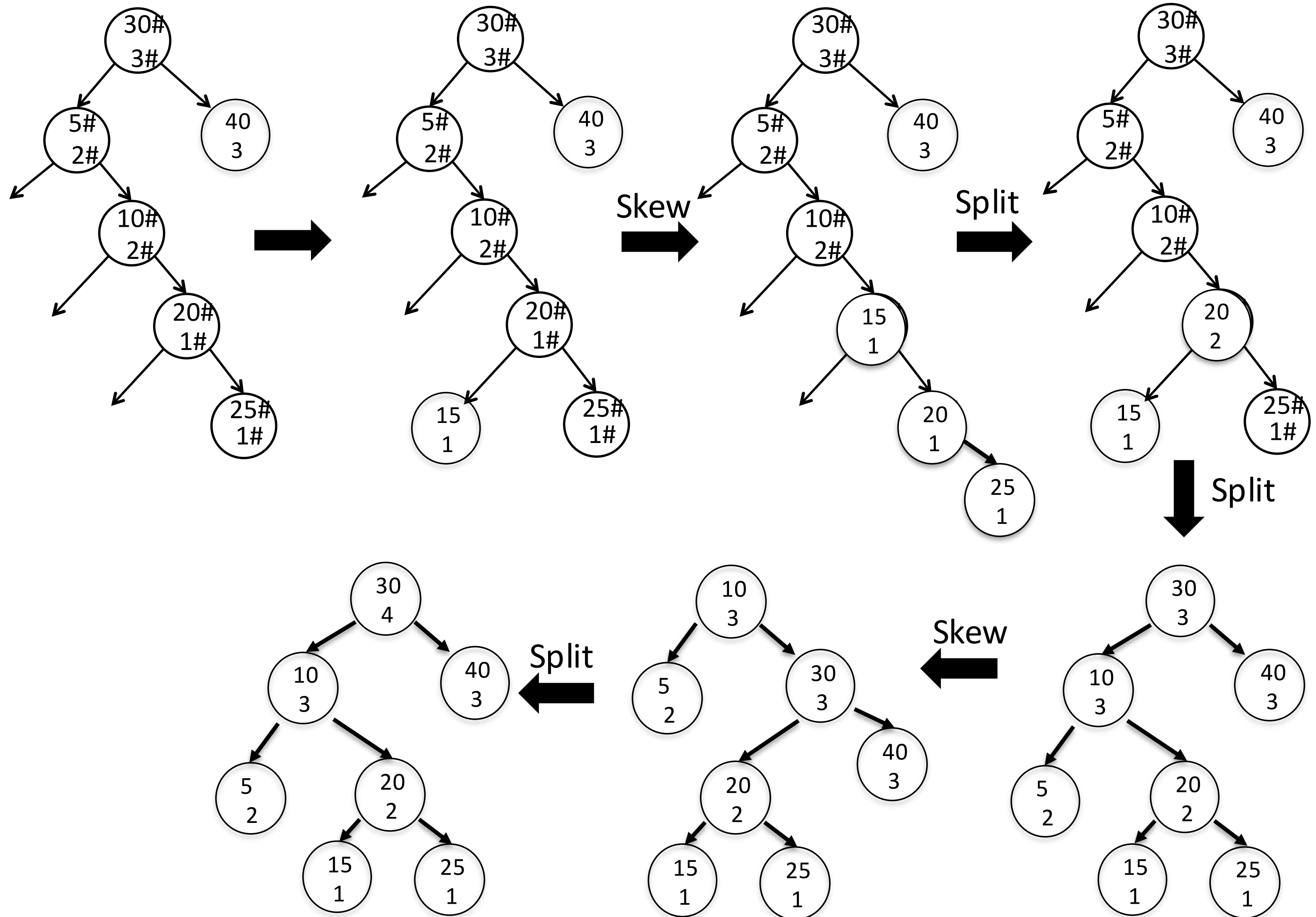
- Localiser là où insérer x
- Insérer x dans une nouvelle feuille F de niveau (1)
- niveauPrec = 0
- Tant que niveau(F) != niveauPrec
  - Si F est enfant gauche de son parent
    - (a) Faire un **skew** (rotation droite) pour aligner F à droite
  - Si F contribue à un ensemble E saturé (i.e. contient plus de **2** éléments)
    - (b) Faire un **split** pour faire monter le milieu de E
      - Faire une rotation gauche pour ramener le milieu au centre
      - F = milieu de E ;
      - niveau(F)++
  - niveauPrec++



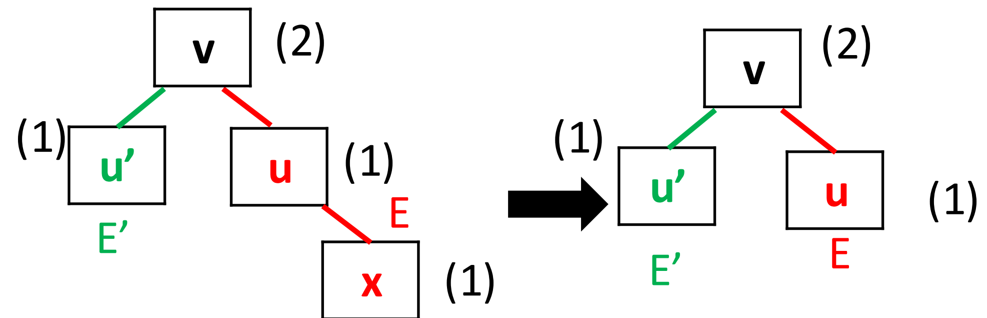
Exemple: Insertion de 15 dans le AA-arbre



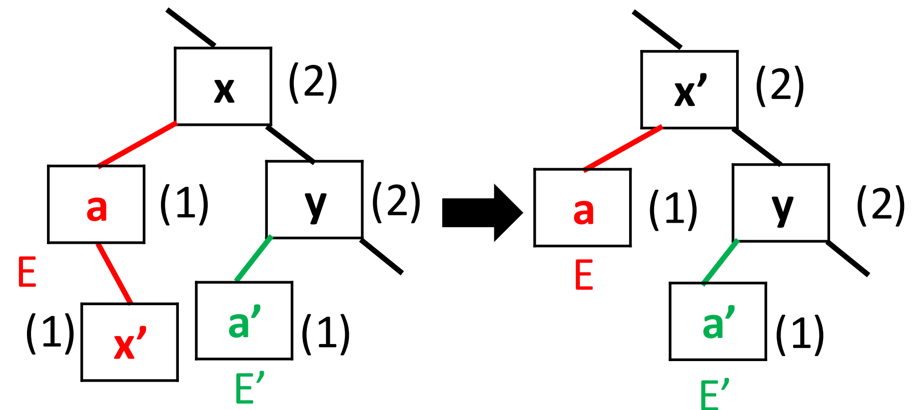
Exemple: Insertion de 15 dans le AA-arbre



Suppression de x dans un AA-arbre



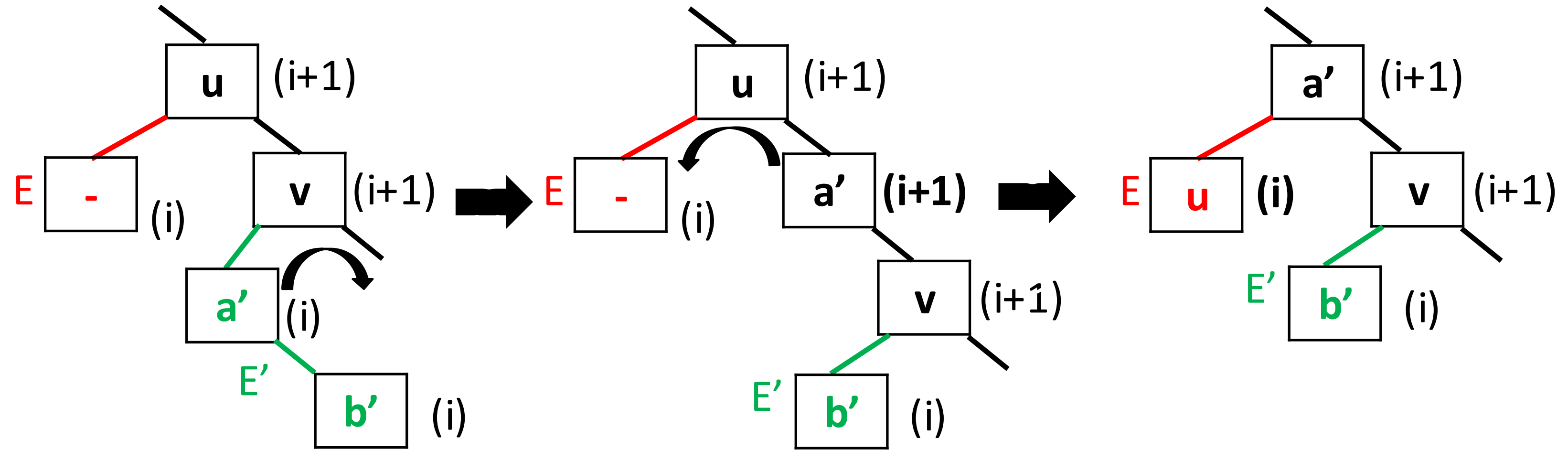
- Localiser le nœud N où se trouve x
- Si N est une feuille
  - E = ensemble auquel contribue N
  - Supprimer N
- Sinon
  - localiser la feuille N contenant le précédent ou le suivant x' de x
  - Échanger x et x'
  - E = ensemble auquel contribue N
  - Supprimer N

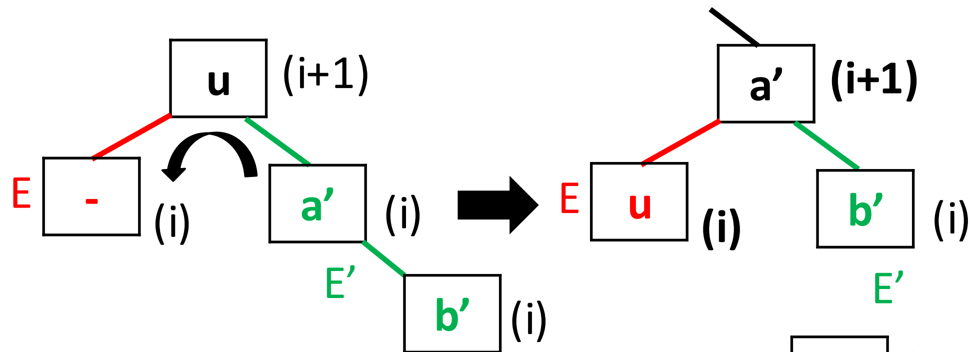


- Tant que E est un ensemble sous-rempli (i.e. contient 0 éléments):
  - (a) (Tranfert) Si le frère adjacent E' de E a plus de 1 élément (i.e 2 éléments)
    - Transférer un élément un élément du parent vers E et de E' vers l'ensemble parent (rotation + incrémentation et décrémentation de niveaux)
  - (b) (Fusion) Sinon (i.e E' a 1 élément):
    - Fusionner de E avec E' + un élément du parent P ((rotations + décrémentations de niveau))
    - E = P

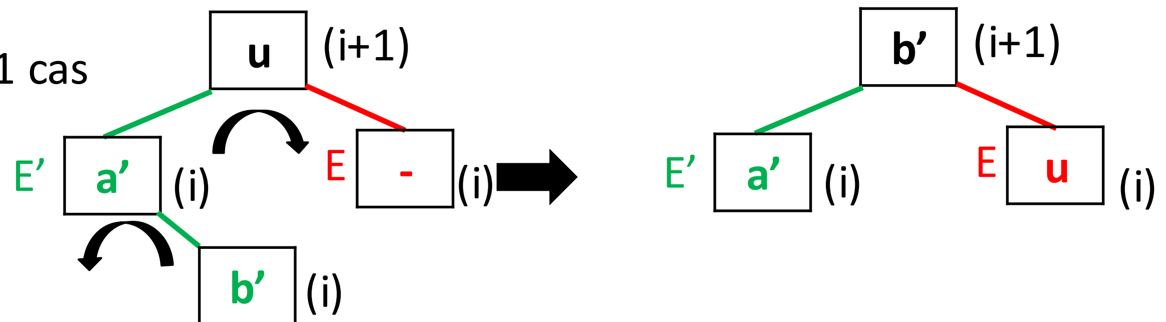
Suppression de x dans un AA-arbre

- (a) Transfert
  - E à gauche de son parent : 2 cas



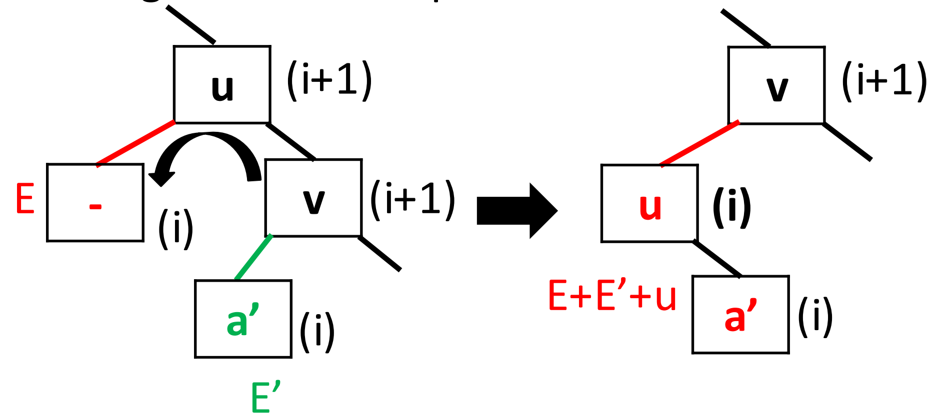


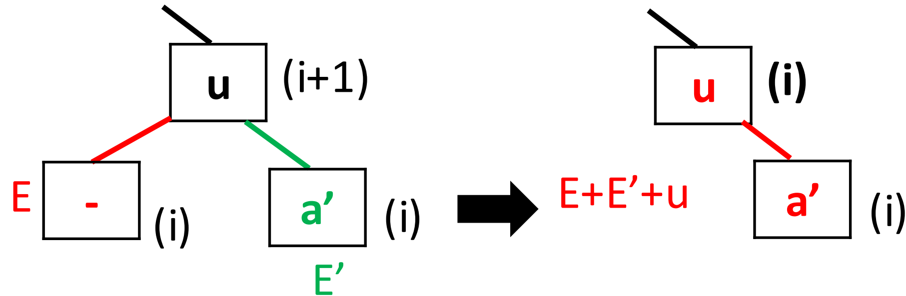
  - E à droite de son parent : 1 cas



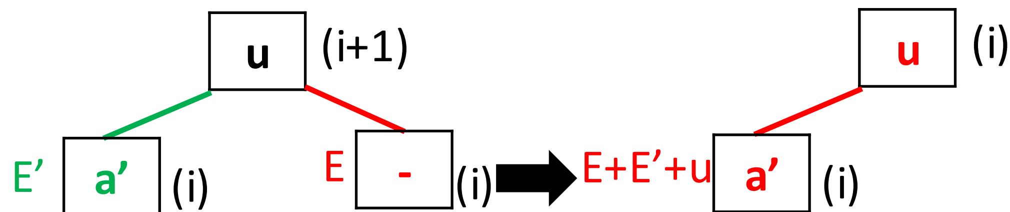
Suppression de x dans un AA-arbre

- (b) Fusion
  - E à gauche de son parent : 2 cas

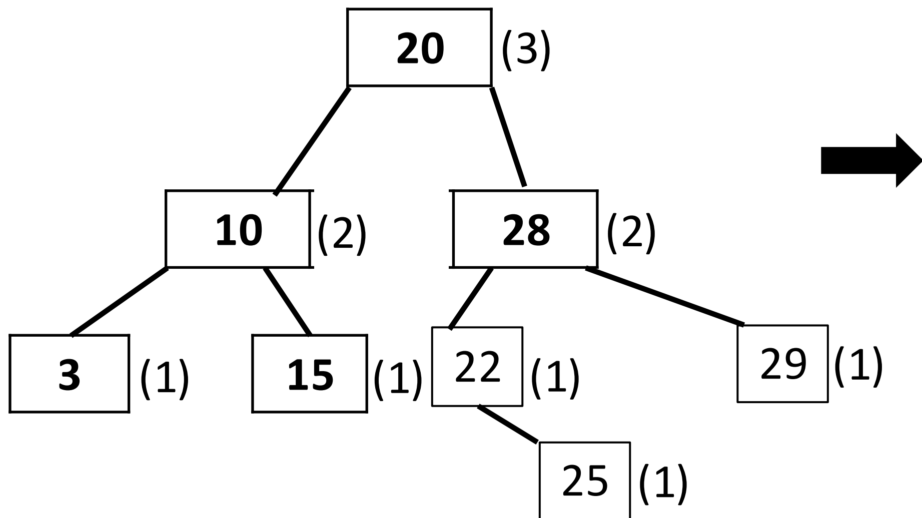




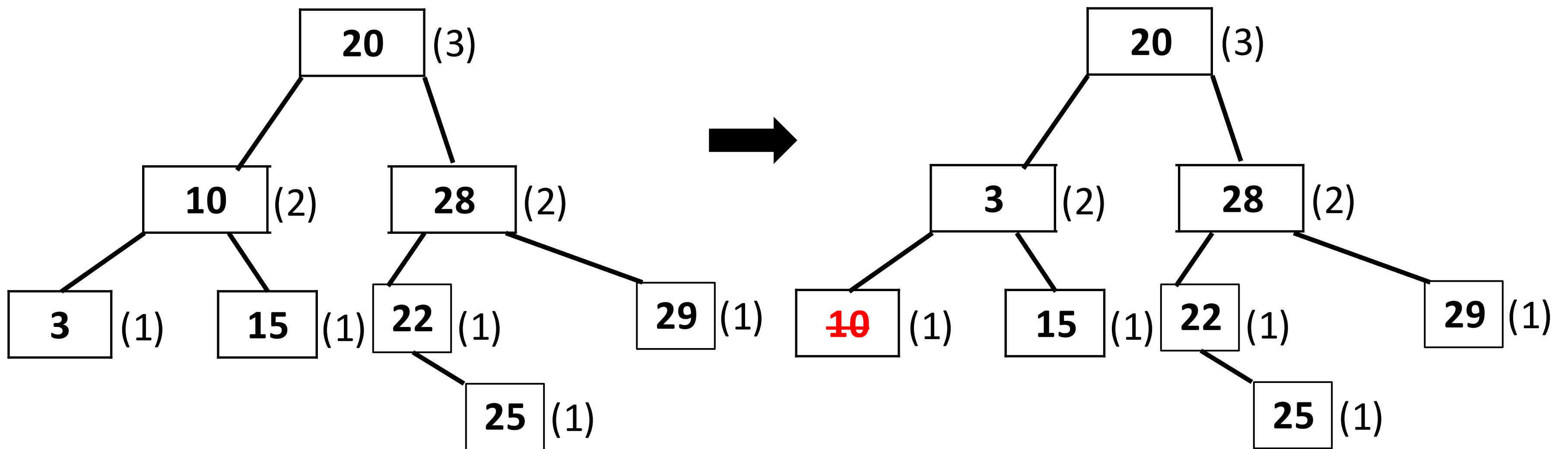
  - E à droite de son parent : 1 cas



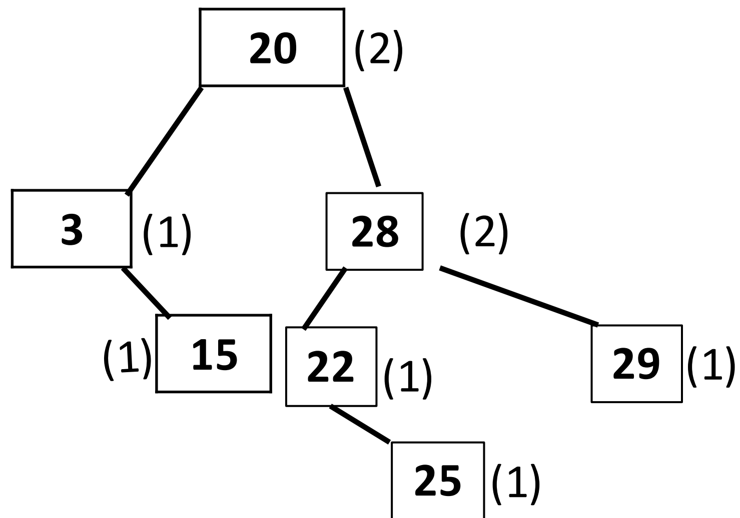
Exemple: Suppresion de 10



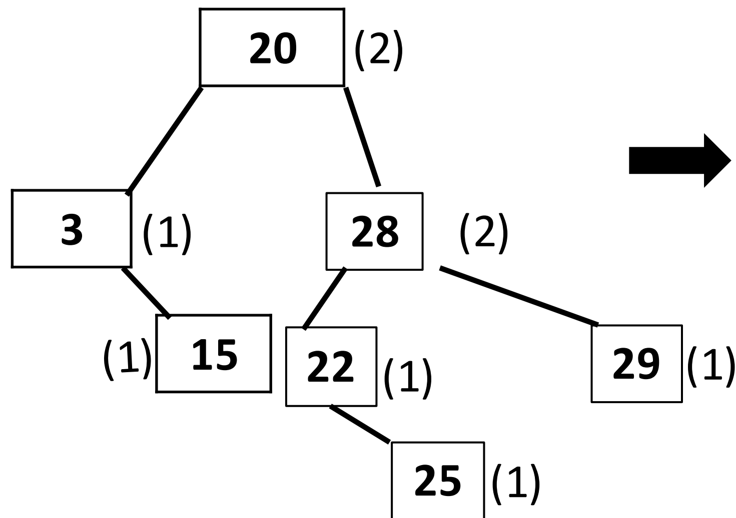
Exemple: Suppresion de 10

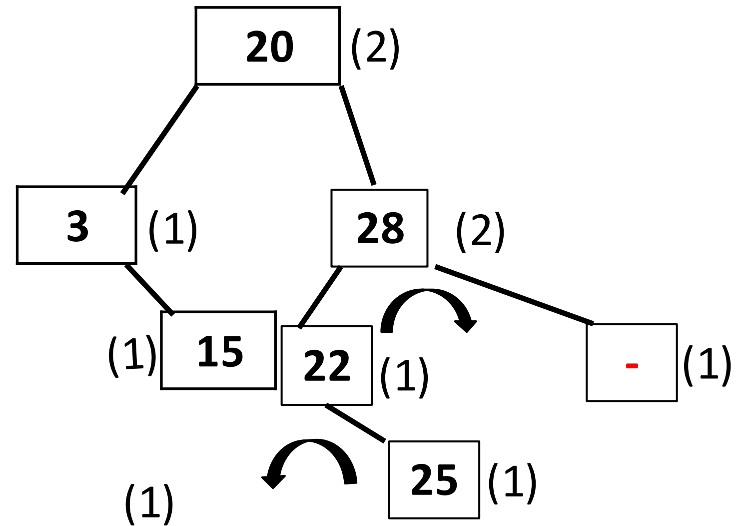


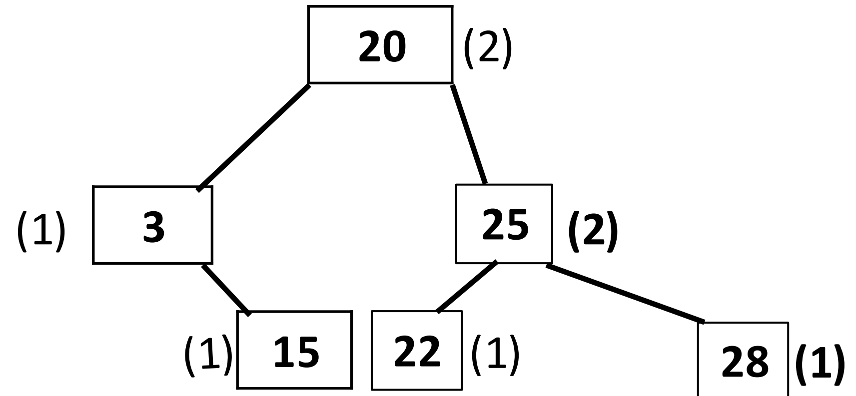
Exemple: Suppresion de 29



Exemple: Suppresion de 29

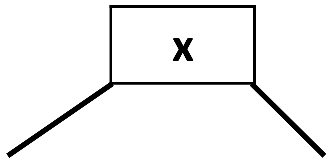


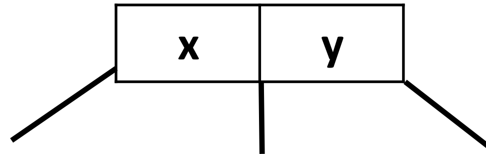


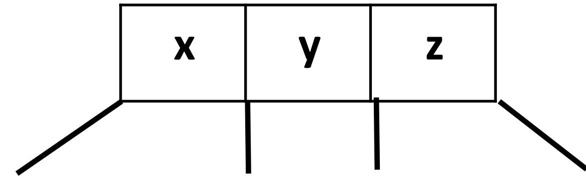


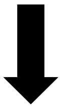
Arbres Rouge et Noir (bicolore)

- Arbre Rouge et Noir : Arbre binaire correspondant à un **B-arbre d'ordre 4**.

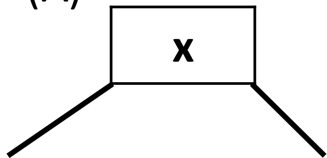




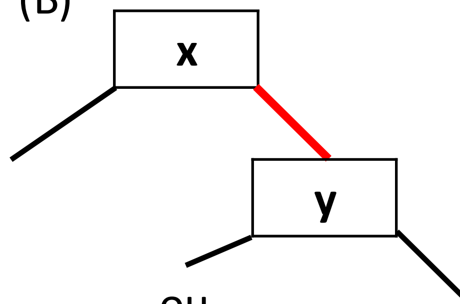




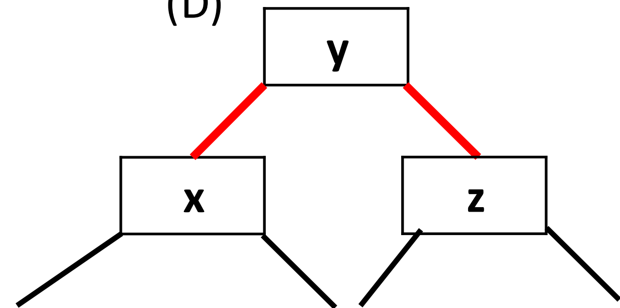
(A)



(B)



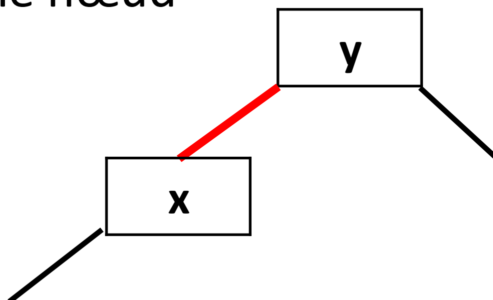
(D)



Arcs rouges : lien entre éléments du même nœud

ou

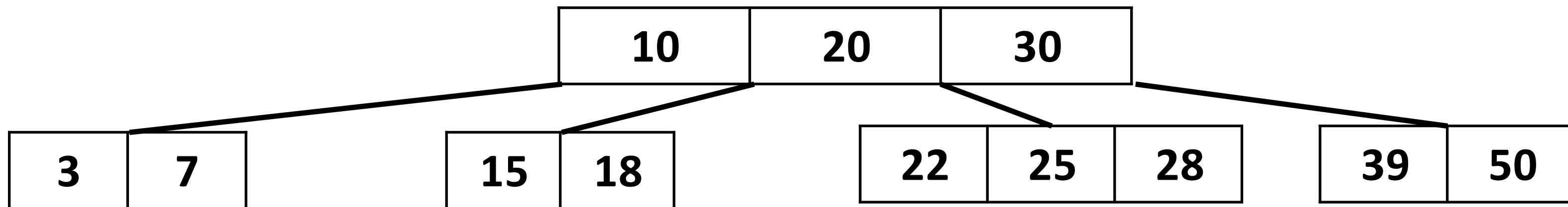
(C)

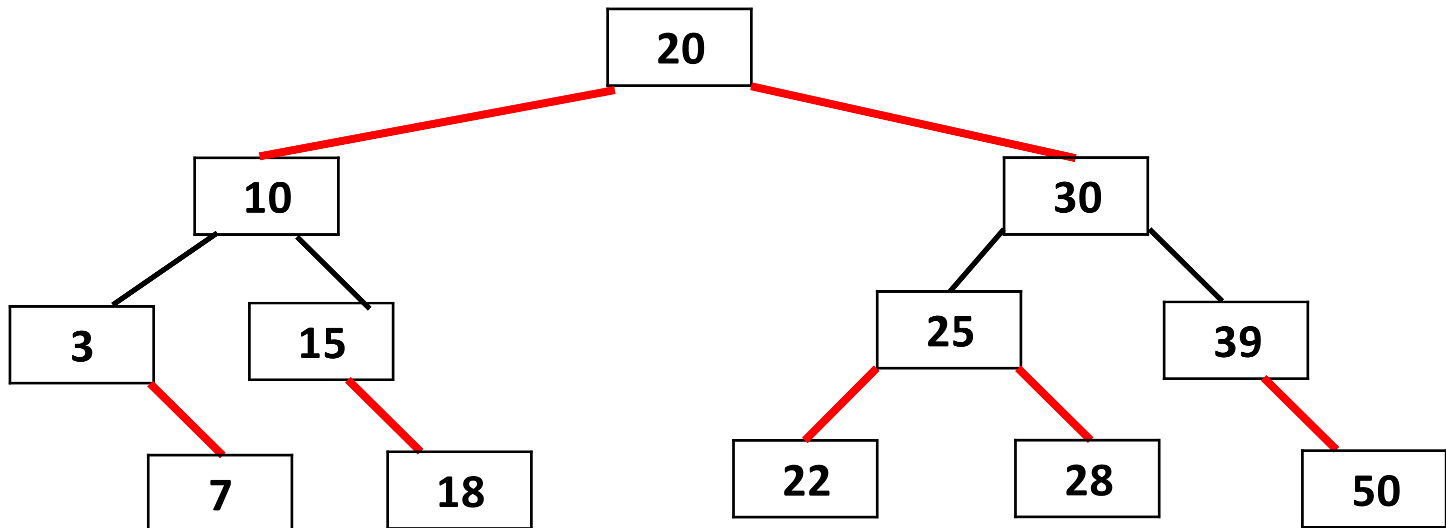


- Tous les chemins racine → feuille ont le même nombre d'arcs noirs
- On n'a jamais 2 arcs rouges de suite

Arbres Rouge et Noir (bicolore)

Exemple

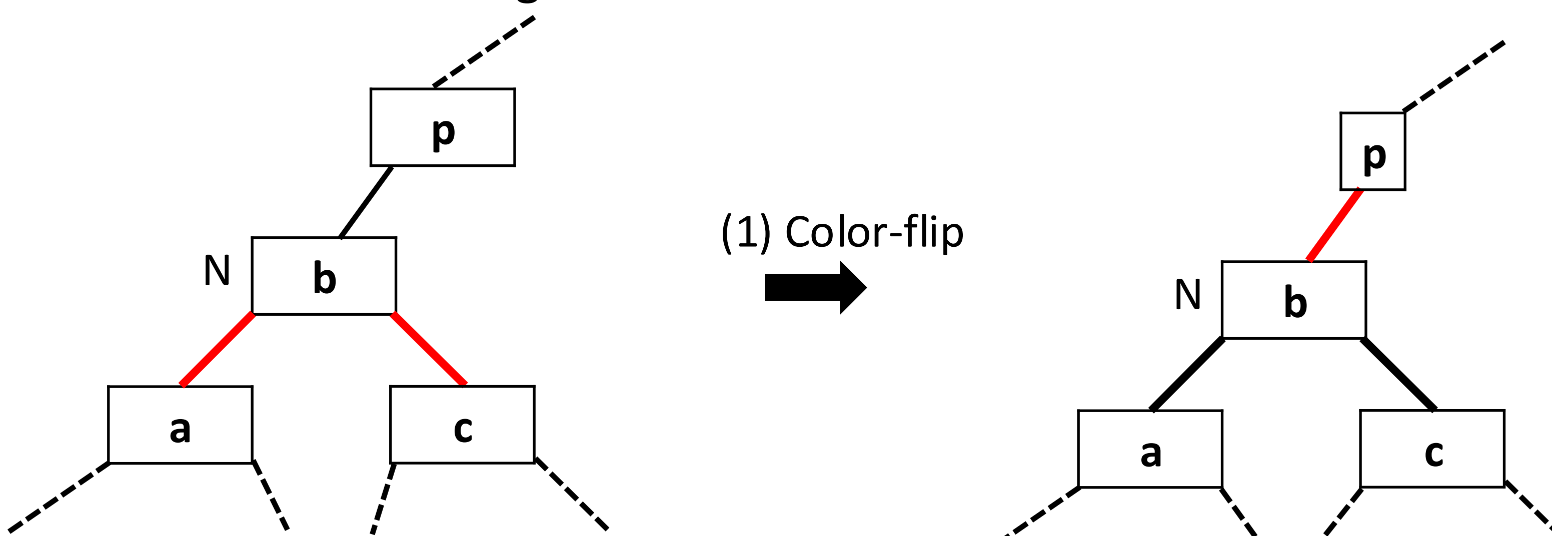




Insertion de x dans un ARN

Idée de la taupe: Ne jamais descendre dans un 3-éléments; On divise les nœuds pendant le parcours avant même de faire l'insertion.

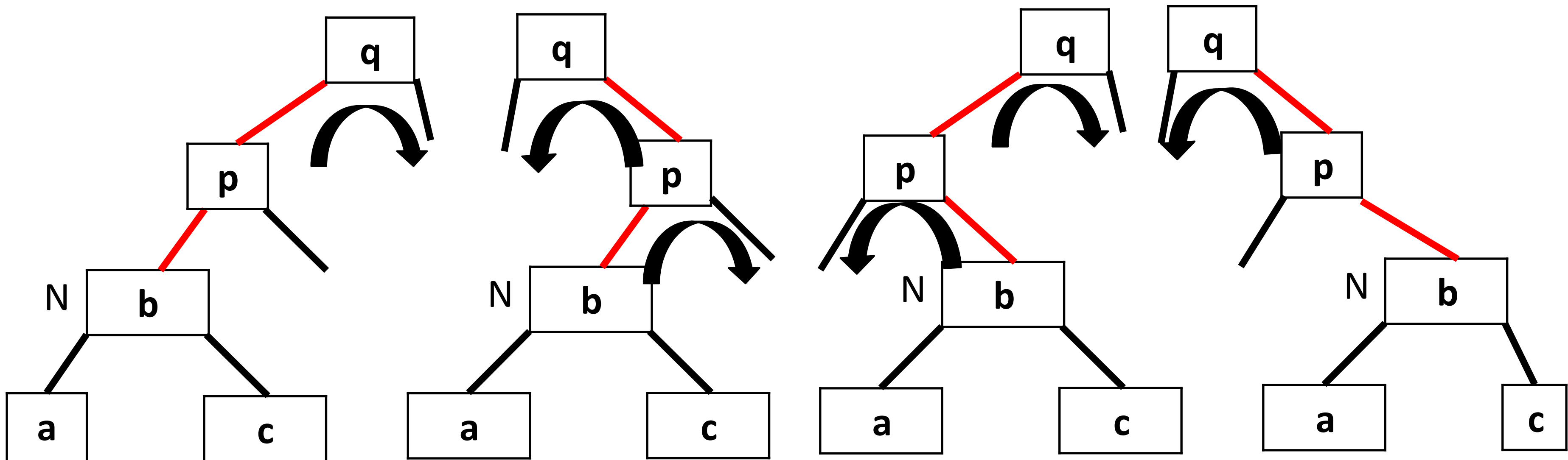
- À partir de la racine, descendre vers la feuille F où insérer x
- Pour chaque nœud N rencontré:
  - Si N est la racine d'un ensemble de 3 éléments:
    - (1) Faire un color-flip sur N pour le faire monter avec son parent
    - (2) Si deux arcs rouges se suivent faire **une ou deux rotations** pour arriver à la configuration (D)
      (rotation pour aligner les deux arcs + rotation pour arriver à (D))
- Insérer x avec un arc rouge



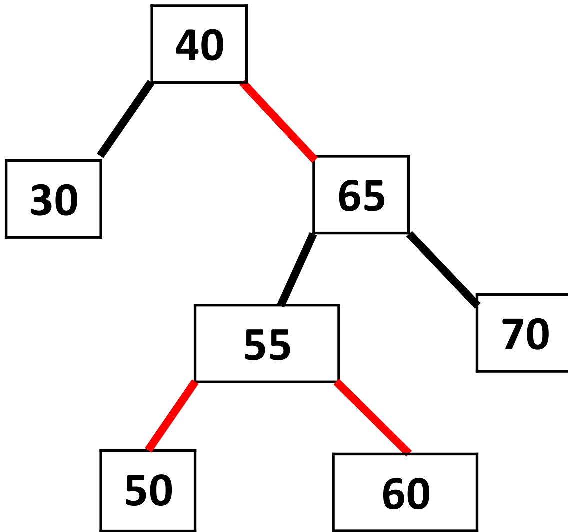
Insertion de x dans un ARN

Idée de la taupe: Ne jamais descendre dans un 3-éléments; On divise les nœuds pendant le parcours avant même de faire l'insertion.

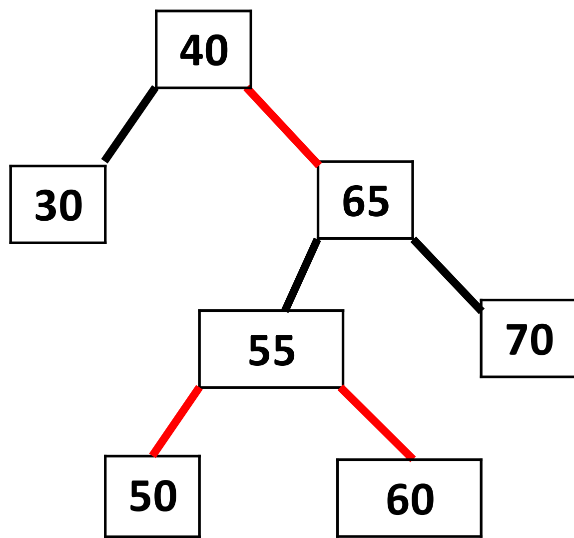
(2) Rotation pour arriver à la configuration (D)



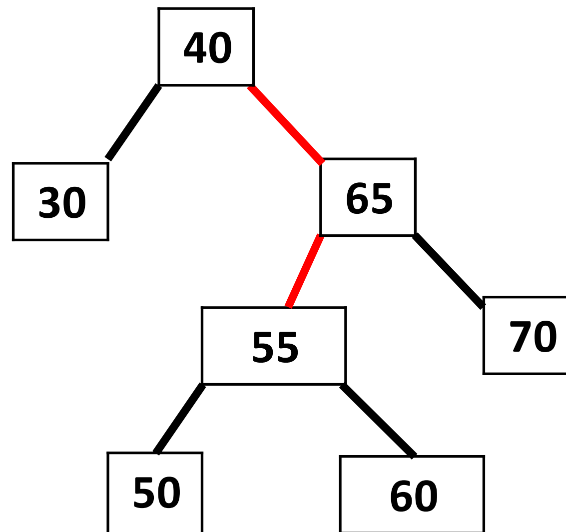
Exemple: Insertion de 52 dans un ARN



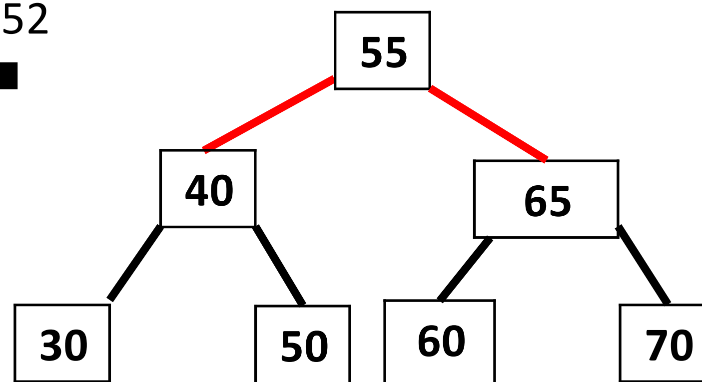
Exemple: Insertion de 52 dans un ARN



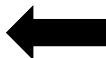
(1) color-flip

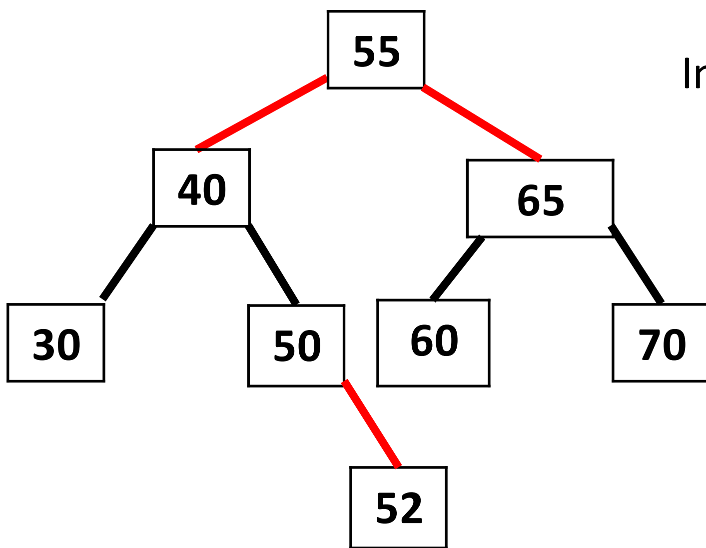


(2) 2 rotations pour arriver à la configuration (D)



Insérer 52





Suppression de x dans un ARN

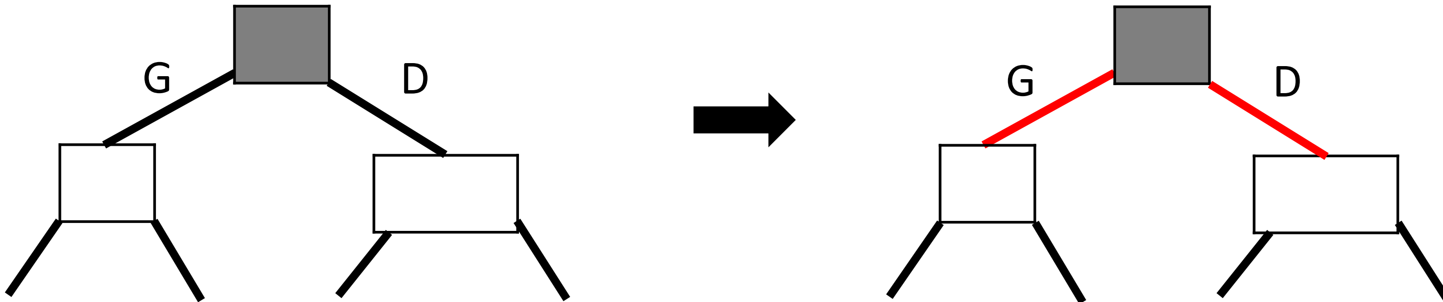
- Même principe que pour tous les arbres binaires de recherche: on se ramène toujours à supprimer une feuille (suppression logique en remplaçant par un autre élément).
- Dans les arbres rouges et noires on doit seulement supprimer des feuilles dont l'arc est rouge ➔ Solution: ne jamais descendre dans un nœud minimal

- Si la racine a deux arêtes G et D noires, et les quatre arêtes descendant de ces deux arêtes sont aussi noires, mettre G et D en rouge
- À partir de la racine, descendre vers la feuille F à supprimer
- Durant la descente, si on doit descendre dans une arête X
  - Si X est rouge, descendre dans X
  - Si X est noire:
    - Cas 1) Si au moins une des deux arêtes descendant de X est rouge, descendre dans X
    - Cas 2) Sinon, si l'arête sœur de X ou l'une de ses deux arêtes descendantes est rouge est rouge, faire une rotation pour avoir une arête rouge (transfert)
    - Cas 3) Sinon, Si l'arête parent est rouge, faire un color-flip, puis descendre
- Supprimer F

Suppression de x dans un ARN

- Même principe que pour tous les arbres binaires de recherche: on se ramène toujours à supprimer une feuille (suppression logique en remplaçant par un autre élément).
- Dans les arbres rouges et noires on doit seulement supprimer des feuilles dont l'arc est rouge.

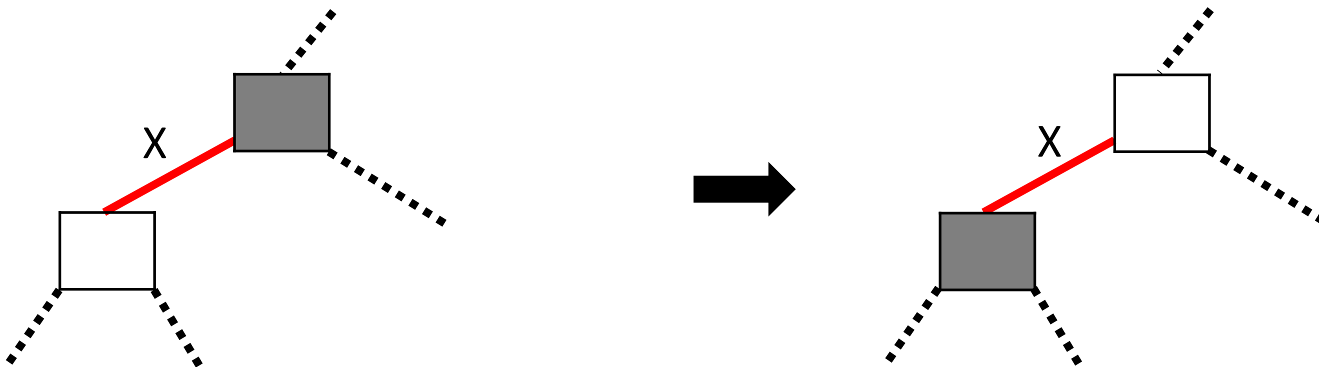
- Si la racine a deux arêtes G et D noires, et les quatre arêtes descendant de ces deux arêtes sont aussi noires, mettre G et D en rouge



Suppression de x dans un ARN

- Même principe que pour tous les arbres binaires de recherche: on se ramène toujours à supprimer une feuille (suppression logique en remplaçant par un autre élément).
- Dans les arbres rouges et noires on doit seulement supprimer des feuilles dont l’arc est rouge.

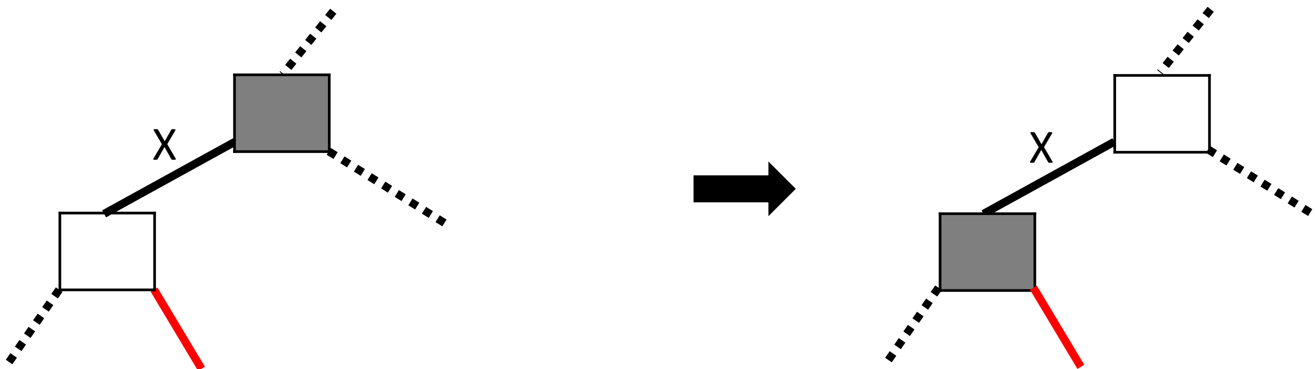
- ❑ Si la racine a deux arêtes G et D noires, et les quatre arêtes descendant de ces deux arêtes sont aussi noires, mettre G et D en rouge
- ❑ À partir de la racine, descendre vers la feuille F à supprimer
- ❑ Durant la descente, si on doit descendre dans une arête X
  - ❑ Si X est rouge, descendre dans X



Suppression de x dans un ARN

- Même principe que pour tous les arbres binaires de recherche: on se ramène toujours à supprimer une feuille (suppression logique en remplaçant par un autre élément).
- Dans les arbres rouges et noires on doit seulement supprimer des feuilles dont l'arc est rouge.

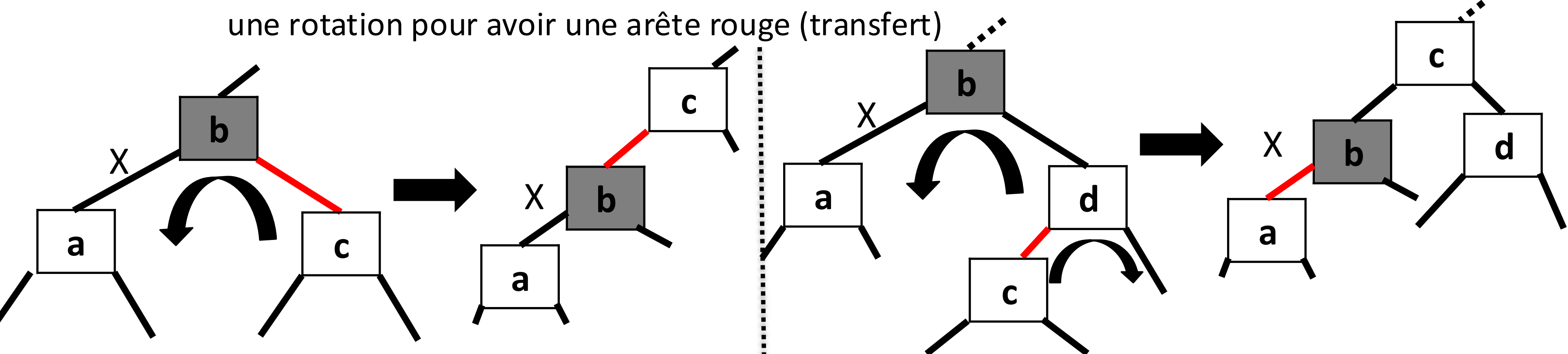
- Si la racine a deux arêtes G et D noires, et les quatre arêtes descendant de ces deux arêtes sont aussi noires, mettre G et D en rouge
- À partir de la racine, descendre vers la feuille F à supprimer
- Durant la descente, si on doit descendre dans une arête X
  - Si X est rouge, descendre dans X
  - Si X est noire:
    - Cas 1) Si au moins une des deux arêtes descendant de X est rouge, descendre dans X



Suppression de x dans un ARN

- Même principe que pour tous les arbres binaires de recherche: on se ramène toujours à supprimer une feuille (suppression logique en remplaçant par un autre élément).
- Dans les arbres rouges et noires on doit seulement supprimer des feuilles dont l'arc est rouge.

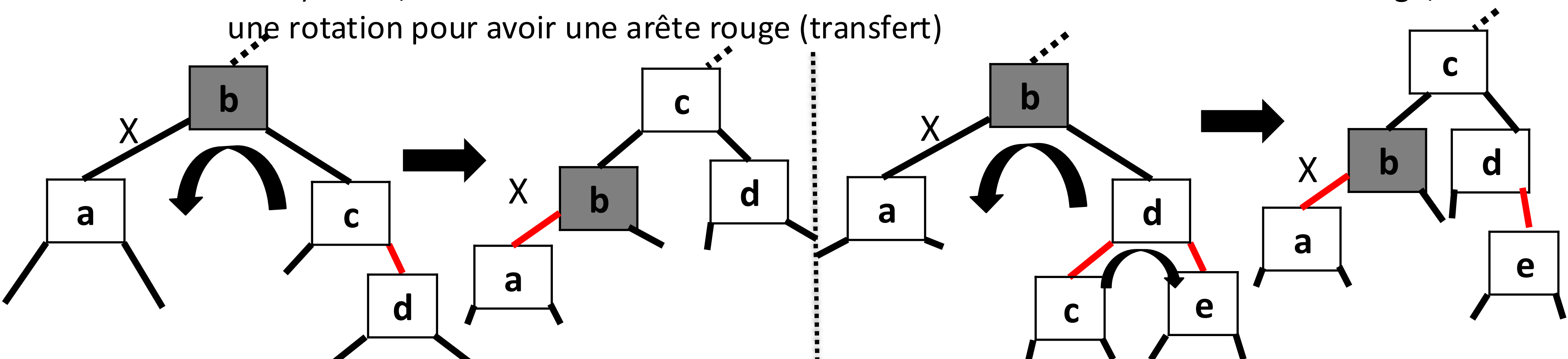
- ❑ Si la racine a deux arêtes G et D noires, et les quatre arêtes descendant de ces deux arêtes sont aussi noires, mettre G et D en rouge
- ❑ À partir de la racine, descendre vers la feuille F à supprimer
- ❑ Durant la descente, si on doit descendre dans une arête X
  - ❑ Si X est rouge, descendre dans X
  - ❑ Si X est noire:
    - ❑ Cas 1) Si au moins une des deux arêtes descendant de X est rouge, descendre dans X
    - ❑ Cas 2) Sinon, si l'arête sœur de X ou l'une de ses deux arêtes descendantes est rouge, faire une rotation pour avoir une arête rouge (transfert)



Suppression de x dans un ARN

- Même principe que pour tous les arbres binaires de recherche: on se ramène toujours à supprimer une feuille (suppression logique en remplaçant par un autre élément).
- Dans les arbres rouges et noires on doit seulement supprimer des feuilles dont l'arc est rouge.

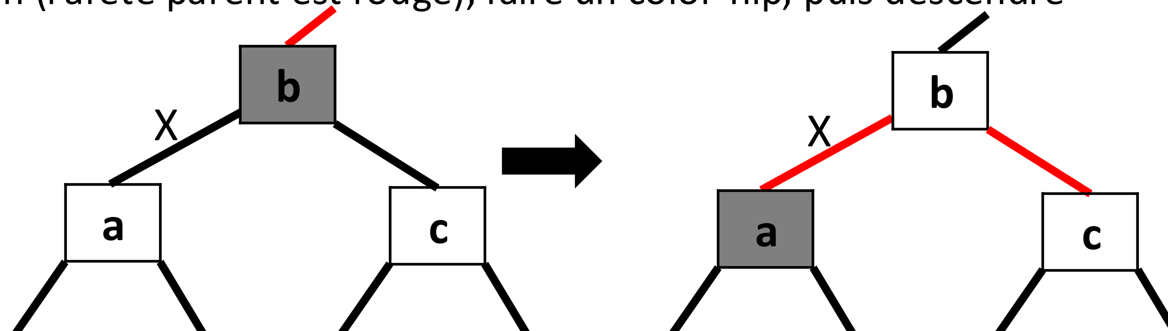
- Si la racine a deux arêtes G et D noires, et les quatre arêtes descendant de ces deux arêtes sont aussi noires, mettre G et D en rouge
- À partir de la racine, descendre vers la feuille F à supprimer
- Durant la descente, si on doit descendre dans une arête X
  - Si X est rouge, descendre dans X
  - Si X est noire:
    - Cas 1) Si au moins une des deux arêtes descendant de X est rouge, descendre dans X
    - Cas 2) Sinon, si l'arête sœur de X ou l'une de ses deux arêtes descendantes est rouge, faire une rotation pour avoir une arête rouge (transfert)



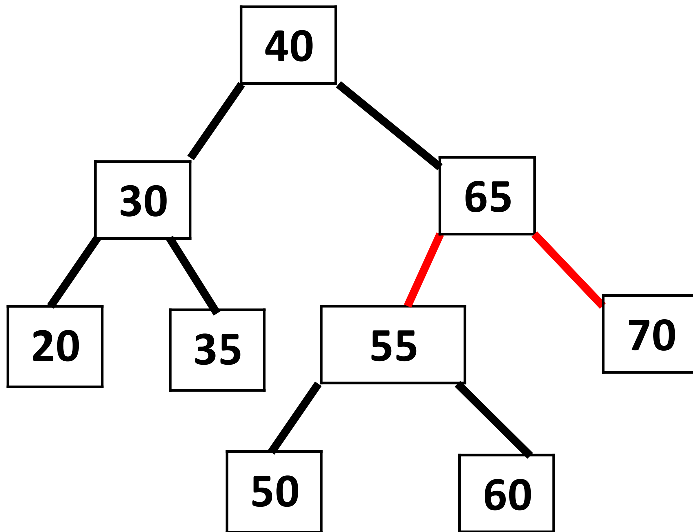
Suppression de x dans un ARN

- Même principe que pour tous les arbres binaires de recherche: on se ramène toujours à supprimer une feuille (suppression logique en remplaçant par un autre élément).
- Dans les arbres rouges et noires on doit seulement supprimer des feuilles dont l'arc est rouge.

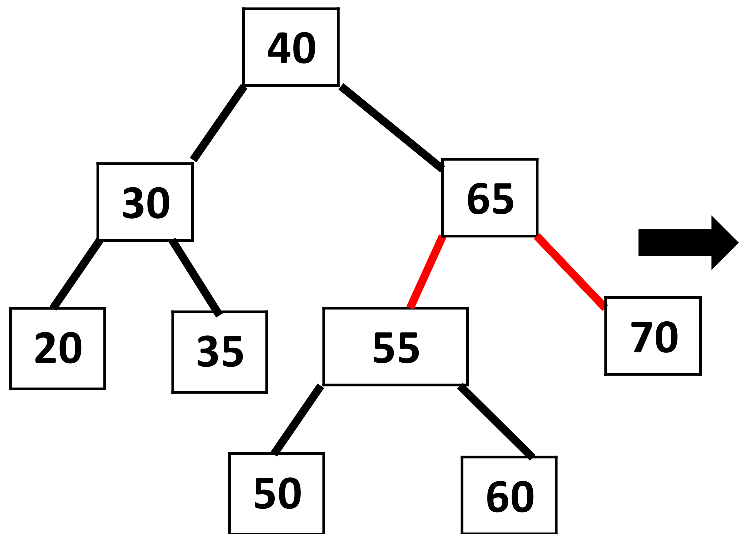
- ❑ Si la racine a deux arêtes G et D noires, et les quatre arêtes descendant de ces deux arêtes sont aussi noires, mettre G et D en rouge
- ❑ À partir de la racine, descendre vers la feuille F à supprimer
- ❑ Durant la descente, si on doit descendre dans une arête X
  - ❑ Si X est rouge, descendre dans X
  - ❑ Si X est noire:
    - ❑ Cas 1) Si au moins une des deux arêtes descendant de X est rouge, descendre dans X
    - ❑ Cas 2) Sinon, si l'arête sœur de X ou l'une de ses deux arêtes descendantes est rouge, faire une rotation pour avoir une arête rouge (transfert)
    - ❑ Cas 3) Sinon (l'arête parent est rouge), faire un color-flip, puis descendre

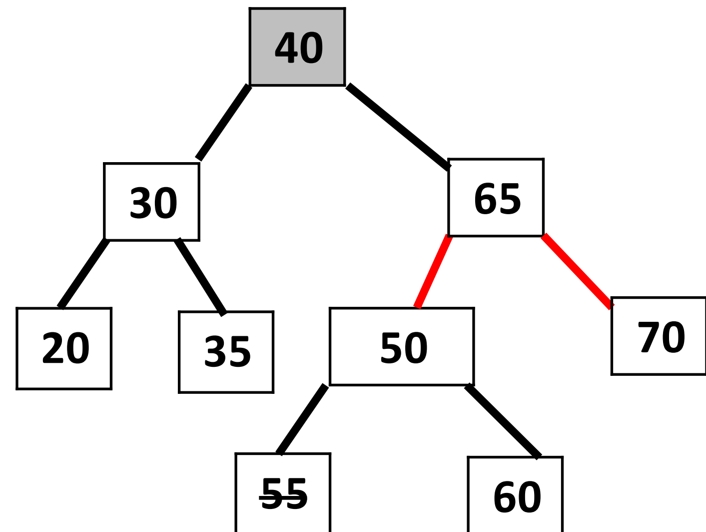


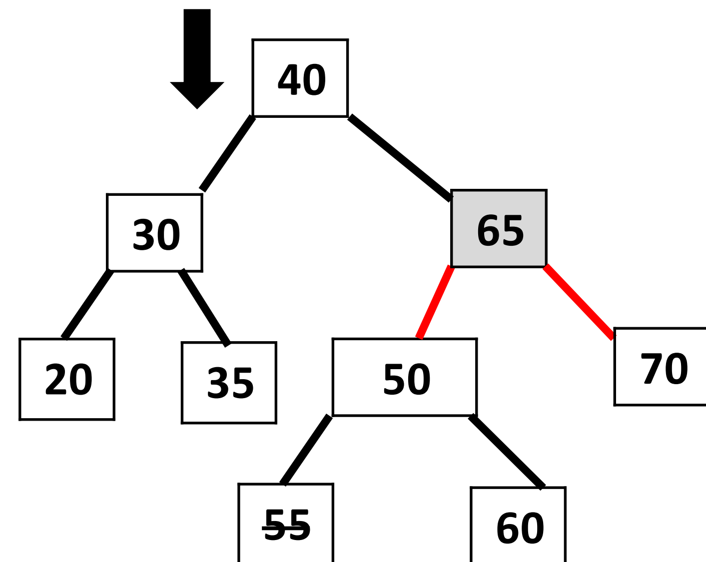
Exemple: Suppression de 55

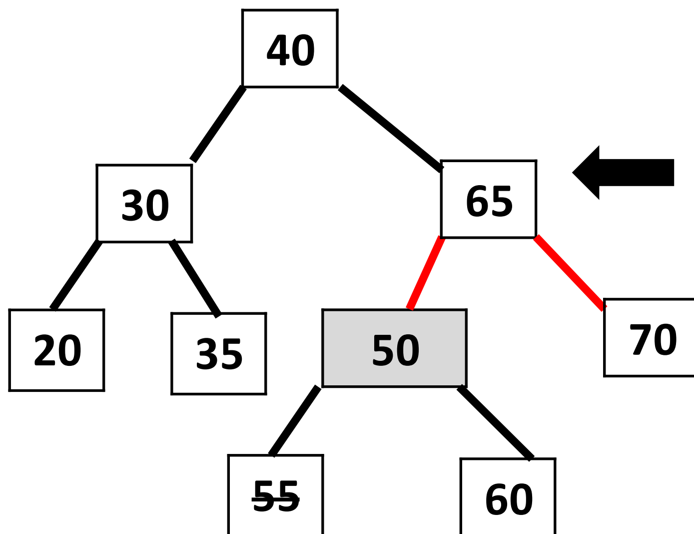


Exemple: Suppression de 55

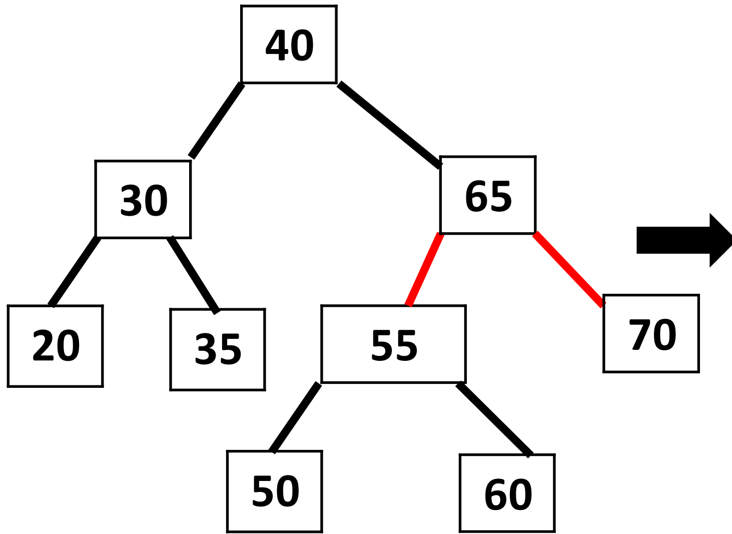


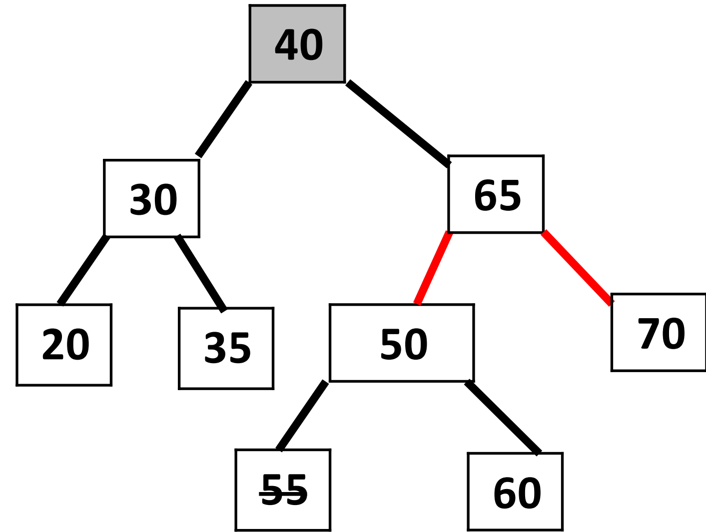


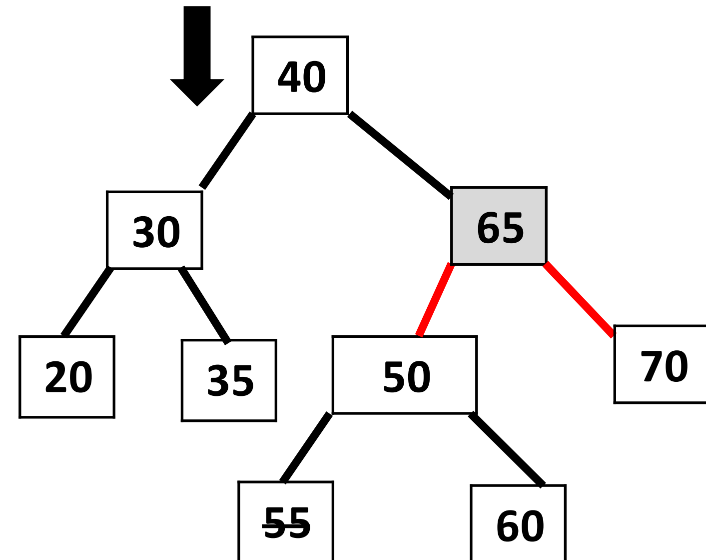


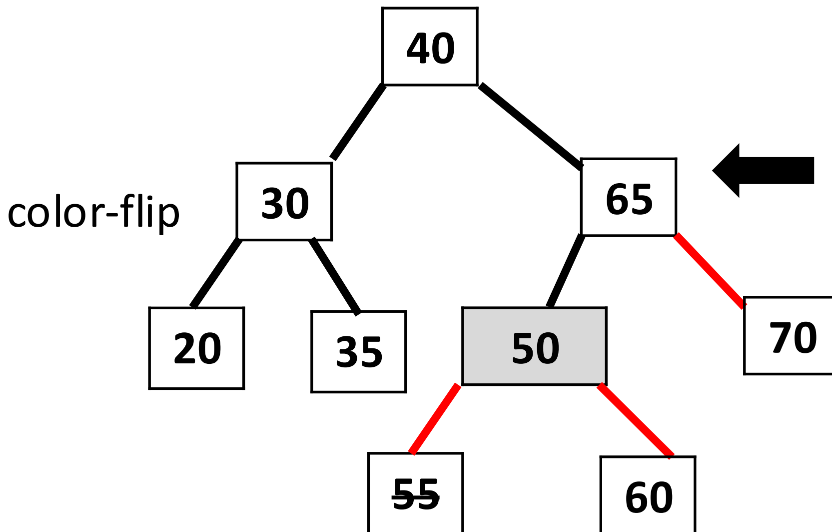


Exemple: Suppression de 55



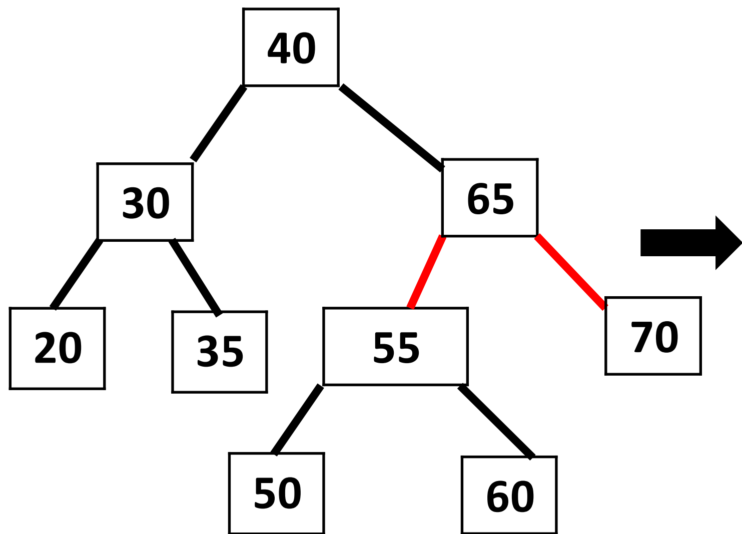


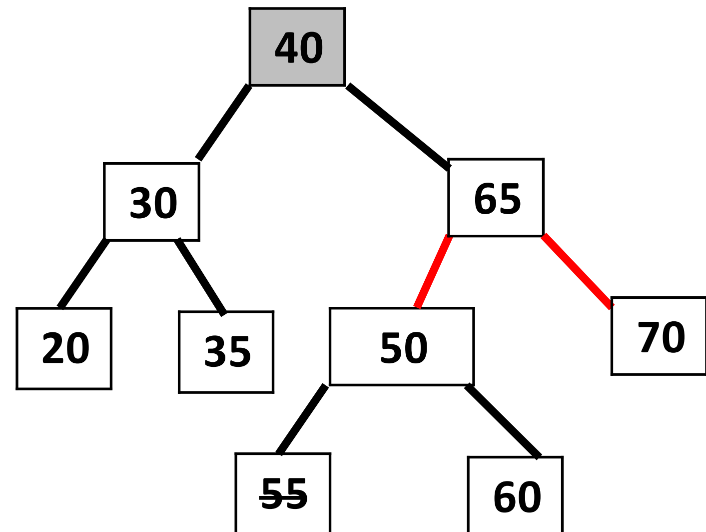


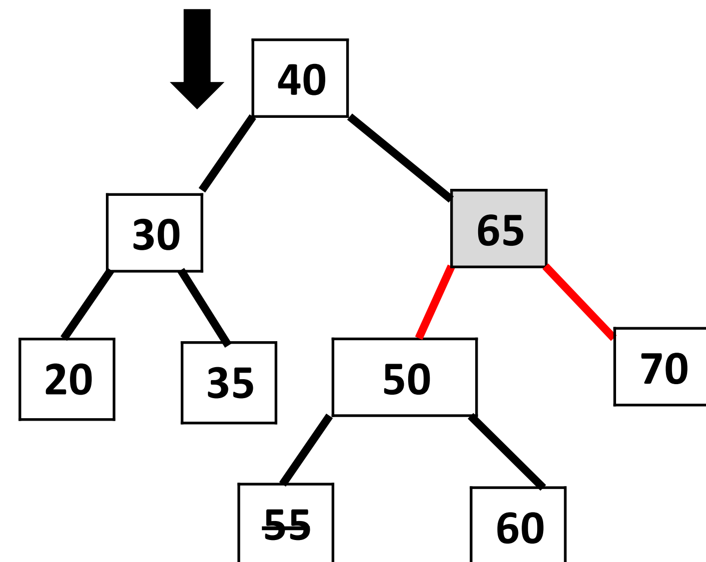


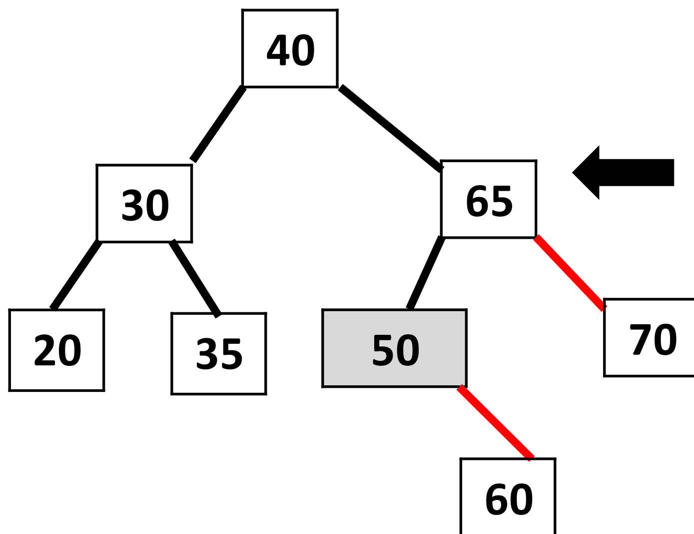
color-flip

Exemple: Suppression de 55

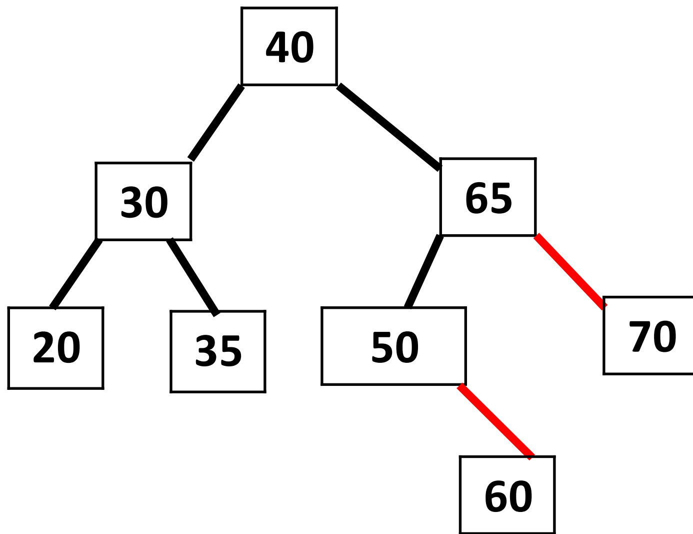




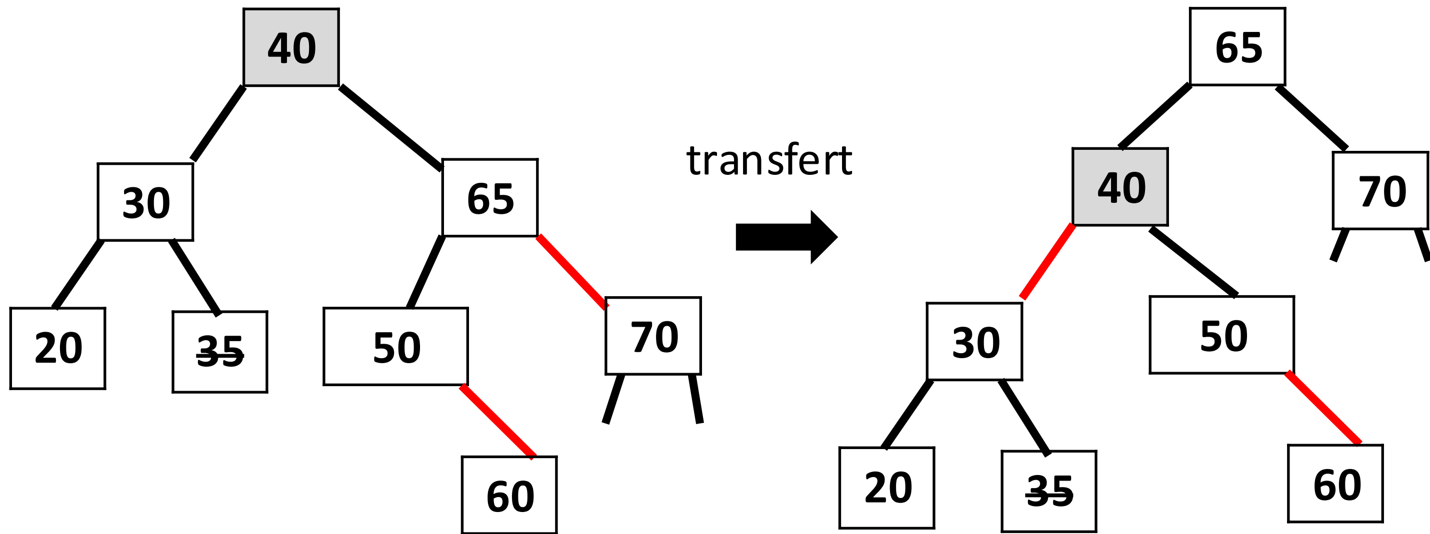


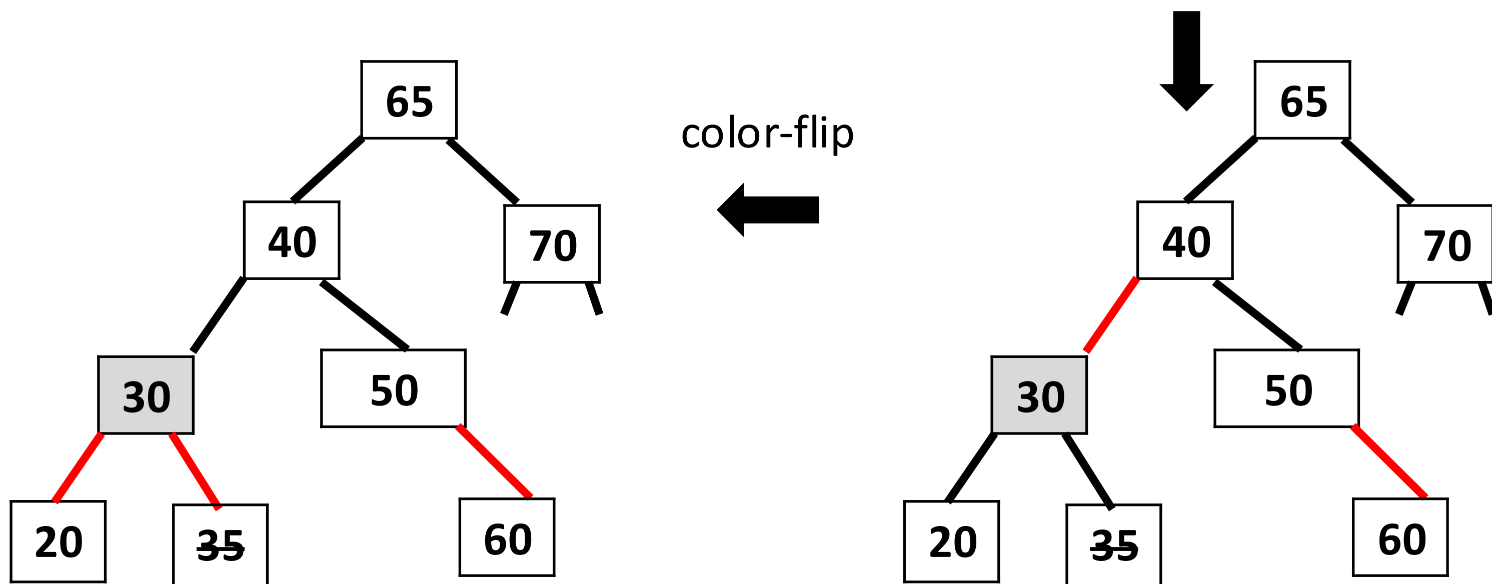


Exemple: Suppression de 35

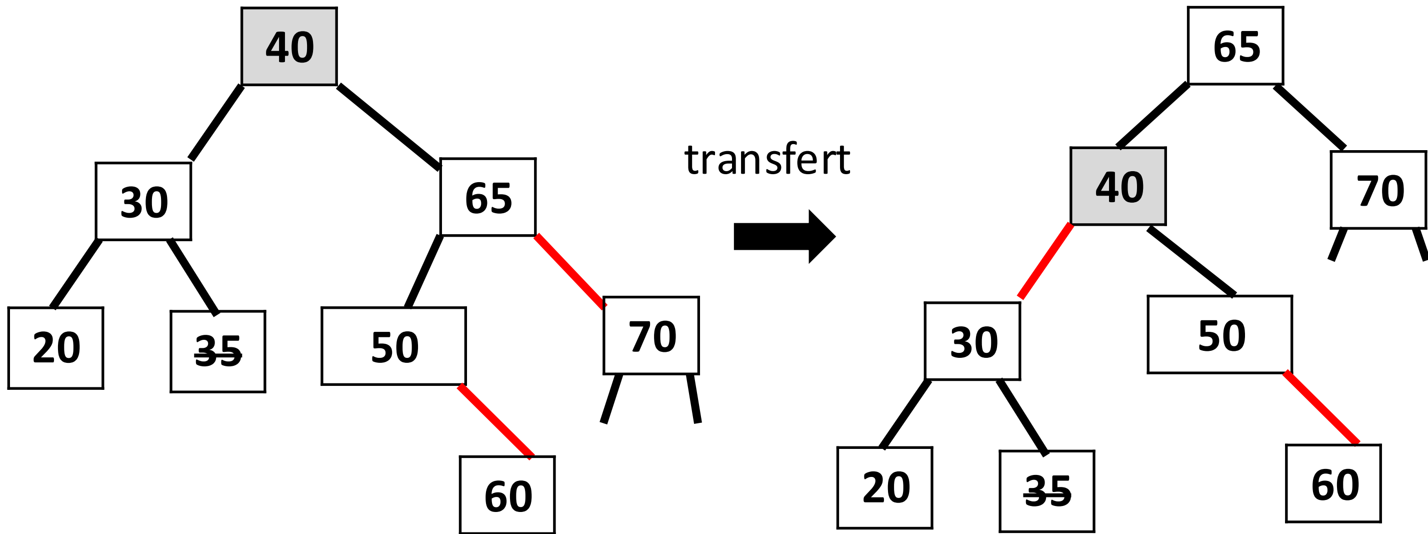


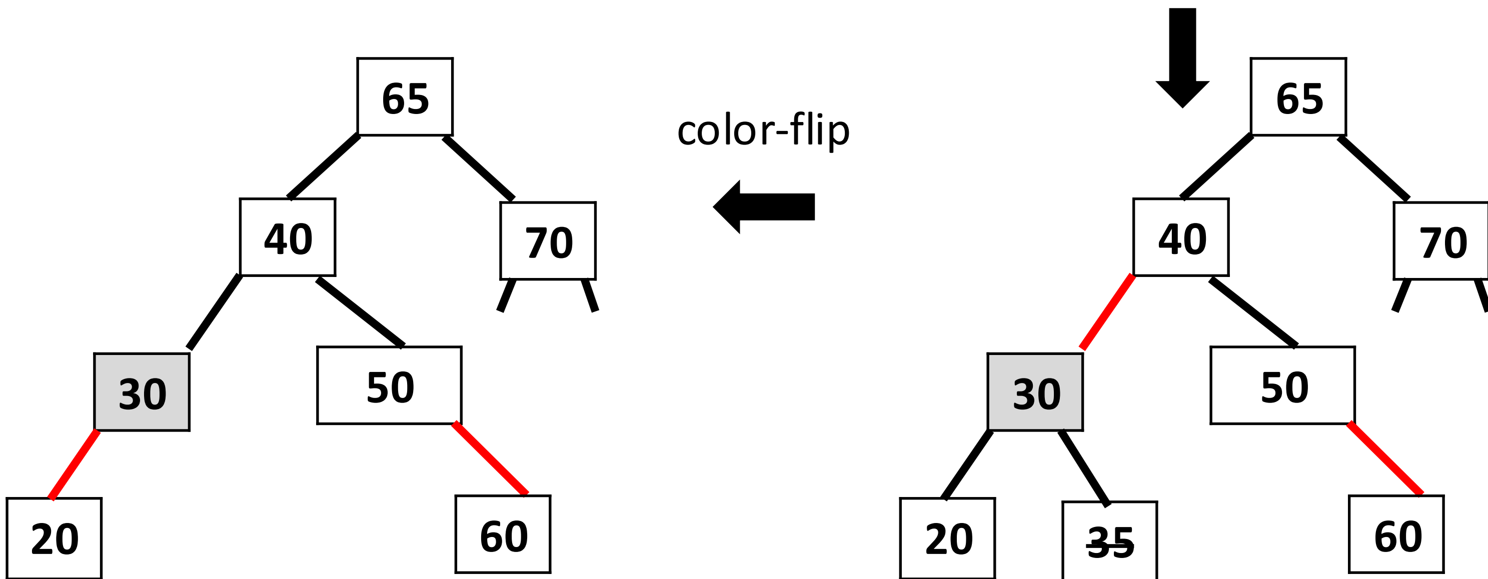
Exemple: Suppression de 35





Exemple: Suppression de 35





Exercice #1: Écrire l’algorithme de l’insert dans un B-arbre pour cette représentation

- Représentation

```
class noeud{
 private:
  deque<type> val;
  nœud* parent;
  deque<nœud*> enfants;
public:
   noeud(type*);
}
```

```
class b-arbre{
 private:
  noeud *racine;
  const size_t m;
  size_t dim;
 public:
  …
}
```

void insert(type x)

- À partir de la racine, descendre vers la feuille F où insérer x
- Pour chaque nœud N rencontré:
  - Si N est plein, i.e N a m-1 éléments:
    - Diviser N en en deux feuilles N1 et N2 telles que N1 contient les (m-1)/2 premiers éléments N2 les (m-1)- (m-1)/2 – 1 derniers et y est l’élément du milieu
    - Insérer y dans le parent de N
- Insérer x dans le dernier nœud (feuille) F

Exercice #1: Écrire l'algorithme de l'insert dans un B-arbre pour cette représentation

- Représentation

```
class noeud{
 private:
  deque<type> val;
  nœud* parent;
  deque<nœud*> enfants;
public:
 noeud(type*);}
```

```
class b-arbre{
 private:
  noeud *racine;
  const size_t m;
  size_t dim;
 public:
  …
}
```

```
void insert(type x)
    nœud* parent = nullptr;
    nœud* n = racine
    size_t rang = 0;
    while(n != nullptr){
            size_t dimn = n->val.size();
            if(dimn == m-1)
                    n = division(n, rang, dimn);
            int i = 0;
            while(n->val[i] < x)
                    i++;
            parent = n;
            n = n->enfants[i];
            rang = i;
    }
     parent->insert(x, rang);
     dim += 1;
```

Exercice #1: Écrire l’algorithme de l’insert dans un B-arbre pour cette représentation

- Représentation

```
class noeud{
 private:
  deque<type> val;
  nœud* parent;
  deque<nœud*> enfants;
public:
 noeud(type*);}
```

```
class b-arbre{
 private:
  noeud *racine;
  const size_t m;
  size_t dim;
 public:
  …
}
```

nœud* division(nœud* n, size_t rang, size_t dimn;)

```
    nœud* n1 = new nœud();
    nœud* n2 = new nœud();
    if(n->parent == nullptr)// cas de la racine
        n->parent = new nœud();
    for(int i =0; i < dimn/2; i++)
        n1->val.push_back(n->val[i]);
        n1->enfants.push_back(n->enfants[i]);
    n1->enfants.push_back(n->enfants[dimn/2]);
    n1->parent = n->parent;
    for(int i = dimn/2+1 =; i < dimn; i++)
        n2->val.push_back(n->val[i]);
        n2->enfants.push_back(n->enfants[i]);
    n2->enfants.push_back(n->enfants[dimn]);
    n2->parent = n->parent;
    n->parent->val.insert(n->val[dimn/2], rang);
    n->parent->enfants[rang] = n1;
    n->parent->enfants.insert(n2, rang+1);
    delete(n)
    return n1->parent;
```

Exercice #2: Écrire l’algorithme de l’erase dans un B-arbre pour cette représentation

- Représentation

```
class noeud{
 private:
  deque<type> val;
  nœud* parent;
  deque<nœud*> enfants;
public:
   noeud(type*);
}
```

```
class b-arbre{
 private:
  noeud *racine;
  const size_t m;
  size_t dim;
 public:
  …
}
```

void erase(type x)

- Localiser le nœud N où se trouve x
- Si N est une feuille
  - F = N; Supprimer x de F
- Sinon
  - localiser la feuille F contenant le précédent x’ de x
  - Échanger les valeurs de x et x’
  - Supprimer x de F
- Tant que F est sous-rempli, i.e. F a moins de m/2 - 1 éléments:
  - (1-Transfert) Si F a un frère adjacent F’ qui a plus de m/2 -1 éléments
    - Transférer un élément de F’ vers le parent et un élément du parent vers F
  - (2-Fusion) Sinon:
    - Fusionner de F avec un un frère adjacent F’ + un élément du parent P
    - F = P

Exercice #2: Écrire l'algorithme de l'erase dans un B-arbre pour cette représentation

❑ Représentation

```
class noeud{
 private:
  deque<type> val;
  nœud* parent;
  deque<nœud*>
enfants;
public:
   noeud(type*);
}
```

```
class b-arbre{
 private:
  noeud *racine;
  const size_t m;
  size_t dim;
 public:
  …
}
```

```
void erase(type x)
   //nœud* n = nœud où l'on supprime une valeur
(initialement feuille);
   size_t rang = rang du nœud n pour son parent;
   while(n->val.size() < m/2-1 and n != racine){
          nœud* ngauche = nulltptr;
          nœud* ndroit = nulltptr;
          if(rang>0)
               ngauche = n->parent->enfants[rang-1];
          if(rang < n->parent->enfants->size()-1)
               ndroit = n->parent->enfants[rang+1];
          if(ngauche != nullptr and ngauche->val.size() >
m/2-1)
               //transfert (ngauche➔parent➔n)
          elif(ndroit != nullptr and ndroit->val.size() > m/2-1)
              //transfert (ndroit➔parent➔n)
          elif(ngauche != nullptr)
              //n = fusion (ngauche + n->parent->val[rang-1]
+ n)
          else
              // n = fusion (n + n->parent->val[rang] + ndroit)
```