Les B-arbres

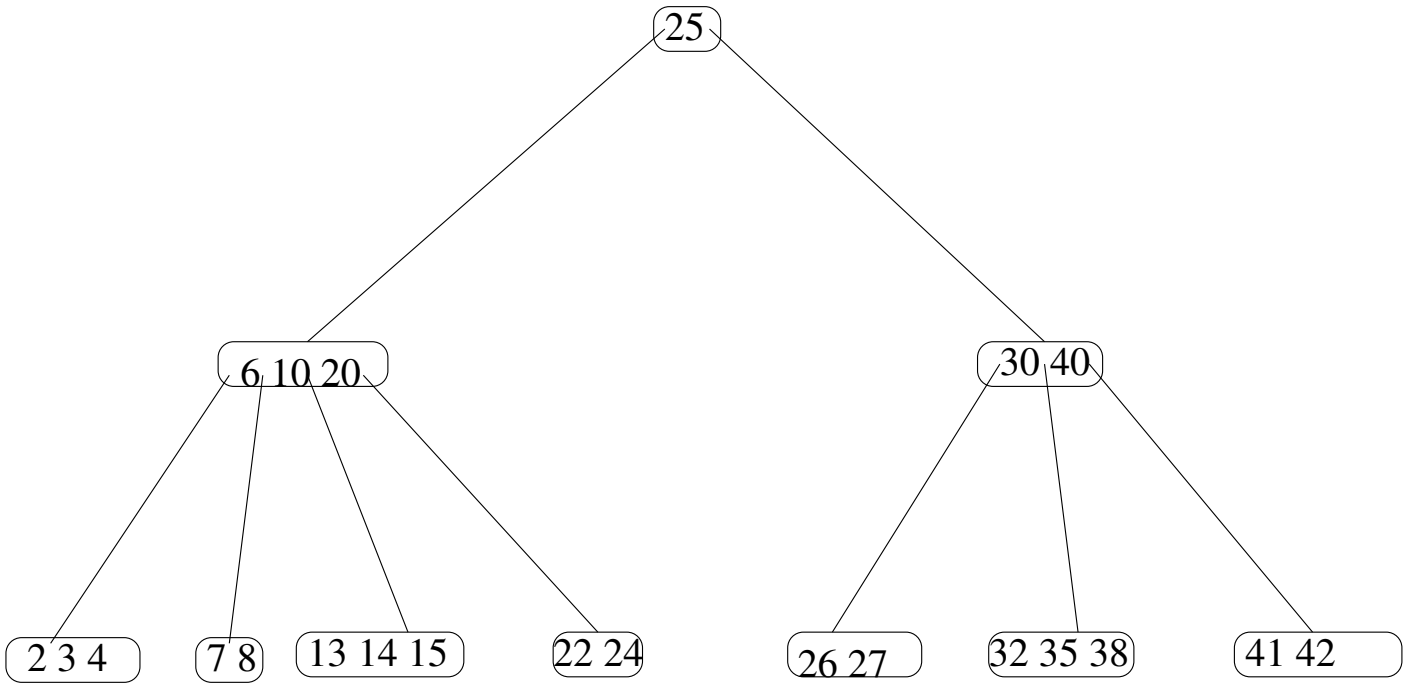
Ce sont des arbres n-aires de recherche, chaque nœud contient plusieurs valeurs ordonnées.
Toutes les feuilles sont à la même hauteur. On controle la hauteur en imposant un nombre minimum d'enfants à chaque nœud (sauf la racine) et en faisant pousser l'arbre par la racine.

1 Définitions

Un B-arbre de degré t (t$\geq$ 2) est un arbre n-aire de recherche avec:

- tous les nœuds, sauf la racine, ont entre t et 2t enfants
- la racine a entre 2 et 2t enfants,(dès que le nombre d'elements dépasse t)
- toutes les feuilles sont à la même hauteur.

Barbre de degré 2

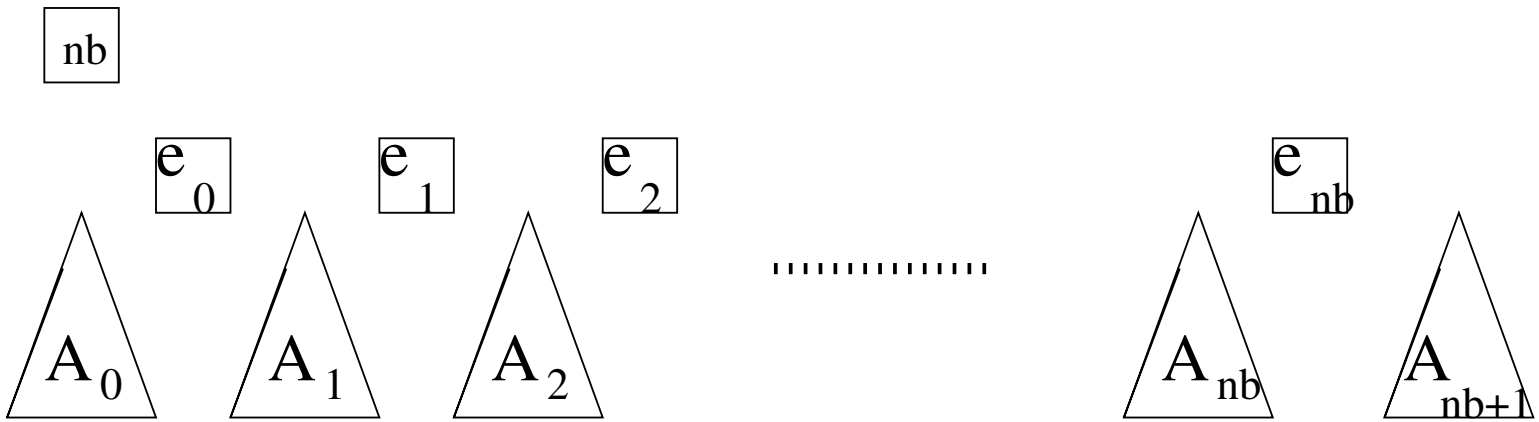


Le nombre minimum de valeurs par nœud est t-1 , le nombre maximum est 2t-1.

La hauteur d'un B-arbre de degré t ayant n nœuds (n ≥ 1 est au plus $\log_t(\frac{n+1}{2})$.
On prend t le plus grand possible, souvent taille d'un nœud$\approx$ taille d'un bloc mémoire, afin de minimiser le nombre d'accès disque.
Nœuds internes de la forme:
$(A_0, e_0, A_1, e_1, \ldots, e_k, A_{k+1})$
A_i sous B-arbre
$k < 2t$
e_i élément vérifiant:
$e_0 < e_1 < \ldots < e_k$
Elément$(A_i) < e_i <$ Elément(A_{i+1})

2 Implantation possible

```
typedef struct bnoeud{
    int nb ; /*nombre de sous arbres*/
    Element val[2*DEGRE-1];
    struct  bnoeud * enfant[2*DEGRE];
} Bnoeud,*Barbre;
/* on pourrait allouer dynamiquement */
```



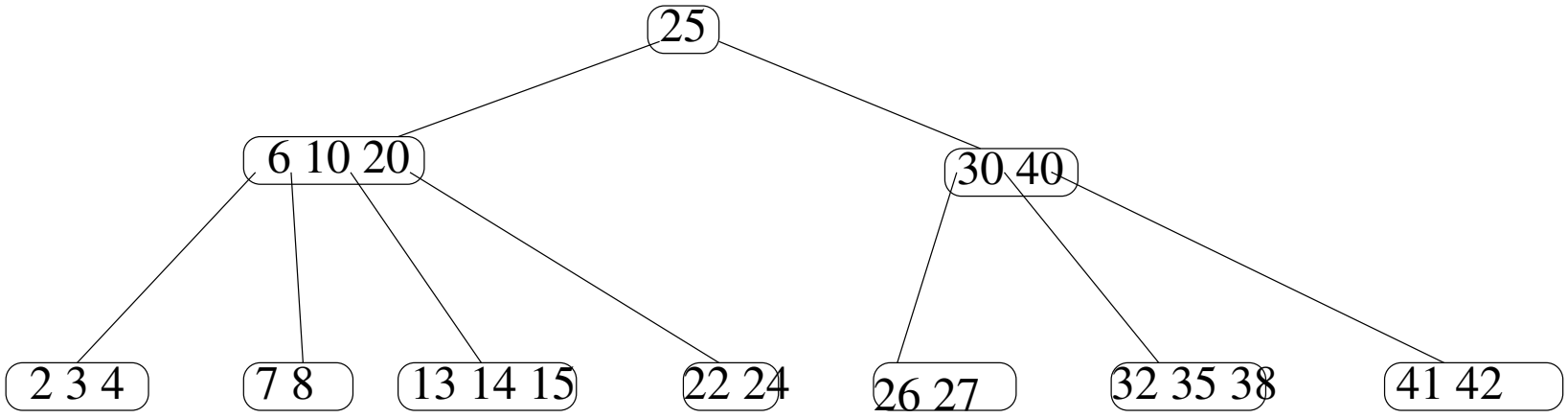
```
int PositionPossible(Element x, Barbre A);
    /* renvoie le plus grand i tel que x>= A.val[i]*/
    /*x == A.val[i] ou x peut etre dans A.enfant[i+1]*/
    /* et -1 si x < A.val[0] x peut etre dans A.enfant[0]*/

int EstElement(Element x,Barbre A){
    int p;
    if(A == NULL) return 0;
    p = PositionPossible(x, A);
    if (p >= 0 && x == A.val[p])
        return 1;
    return EstElement(x, A.enfant[p + 1]);
}
```

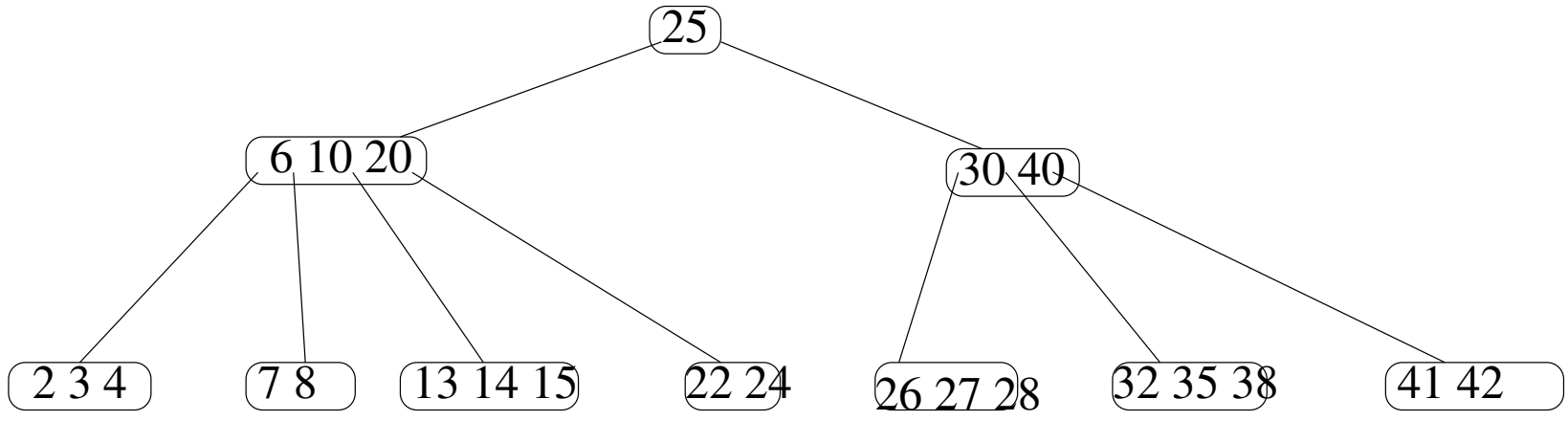
3 Ajout d'un élément x

On utilise la propriété d'arbre de recherche pour trouver la feuille qui contiendra x.

S'il reste de la place dans la feuille,(*moins de* $2t-1$ *éléments*), pas de problème

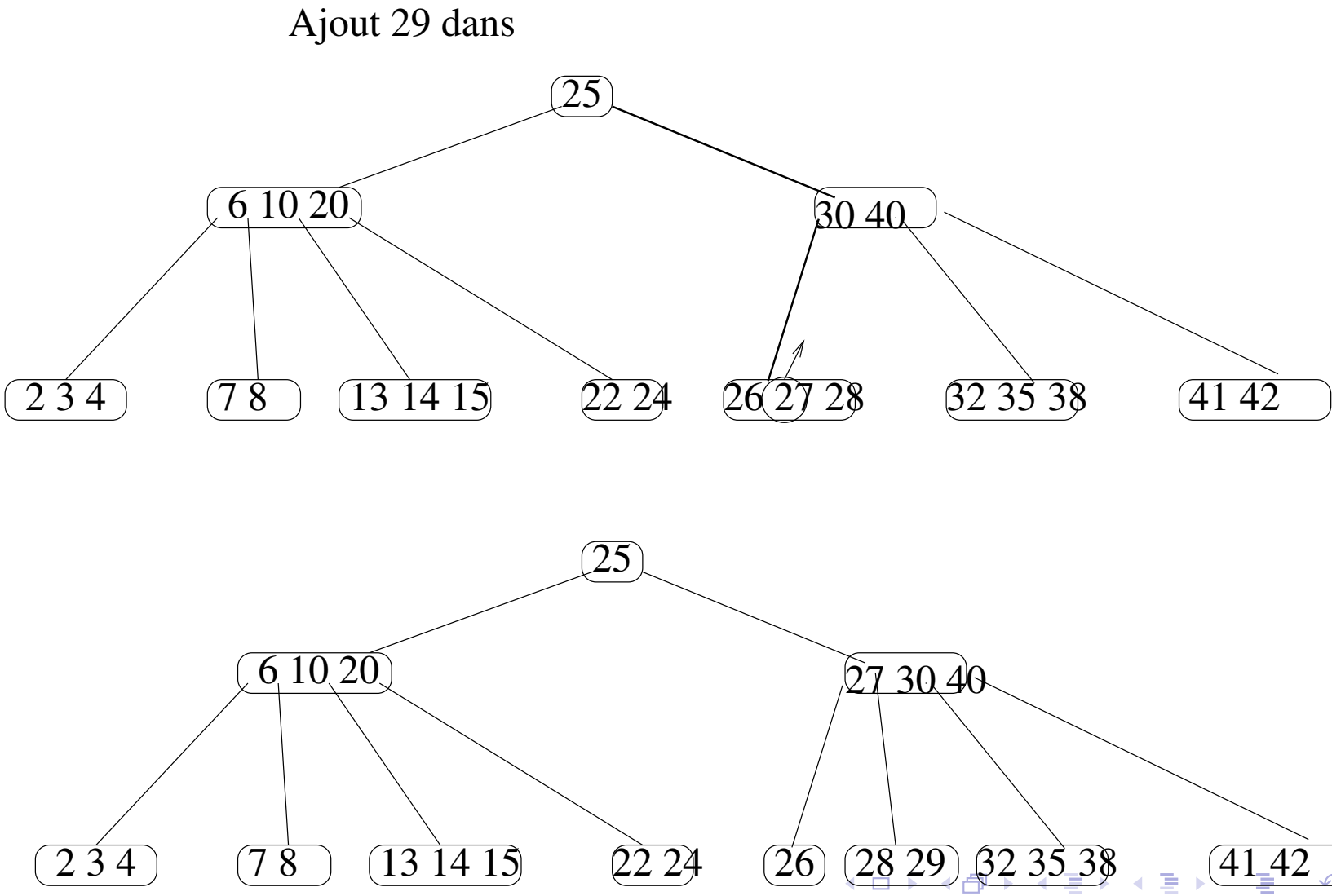


Ajout 28

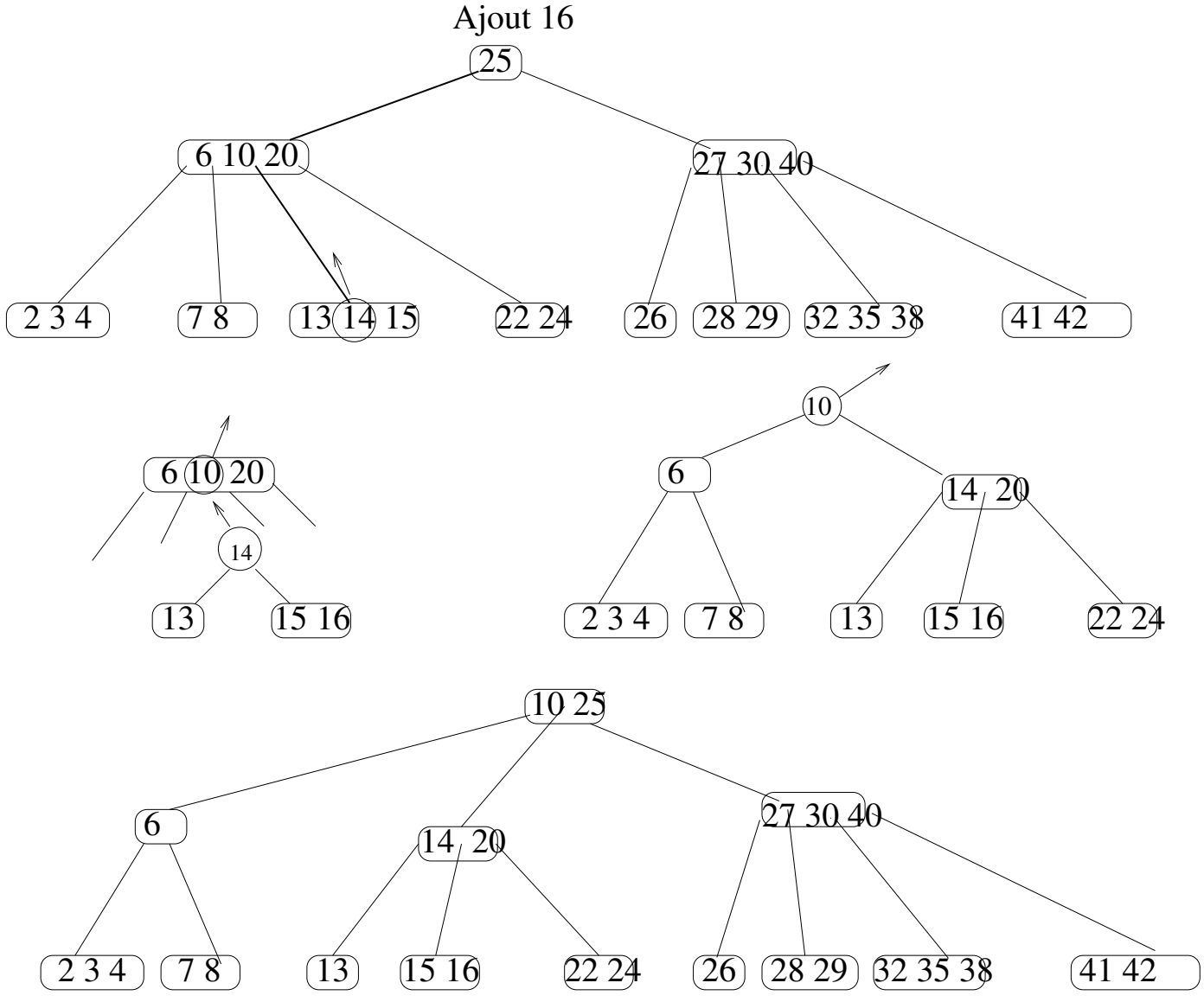


Si la feuille contient déjà $2t - 1$ éléments,

on sépare la feuille en 2 feuilles contenant $t - 1$ éléments autour du médian, le médian remonte pour être inséré dans le nœud parent.
On insère x dans une feuille contenant $t - 1$ éléments,



Cela peut entrainer des séparations de nœuds en cascades si les ancêtres de la feuille à séparer sont des nœuds pleins.



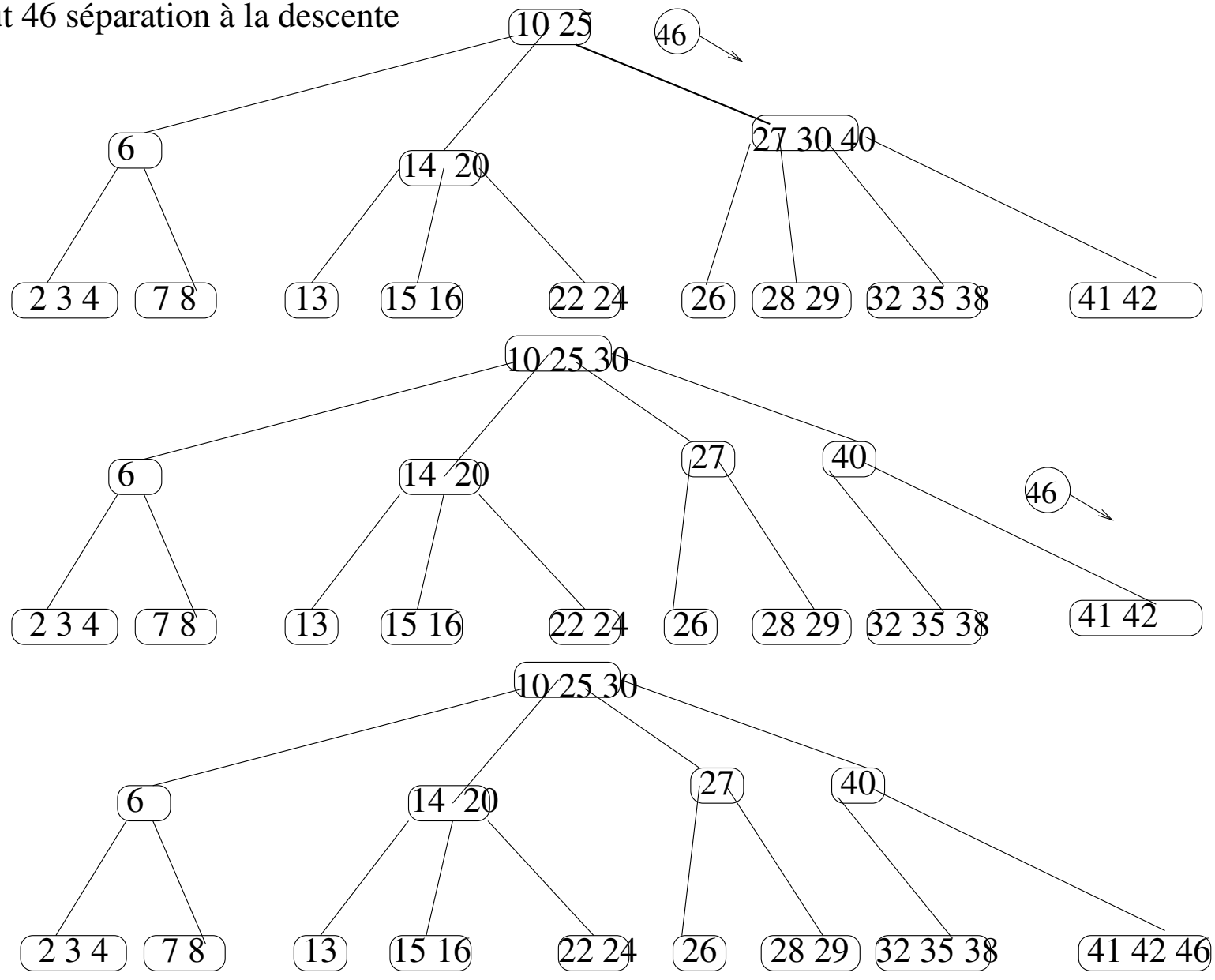
Variante à l'ajout

La possibilité de remontée en cascade interdit l'utilisation simultanée de l'arbre par plusieurs utilisateurs. Il faut mettre un verrou sur un nœud pouvant être changé et donc sur la racine.

La séparation à la descente limite les remontées à un niveau.

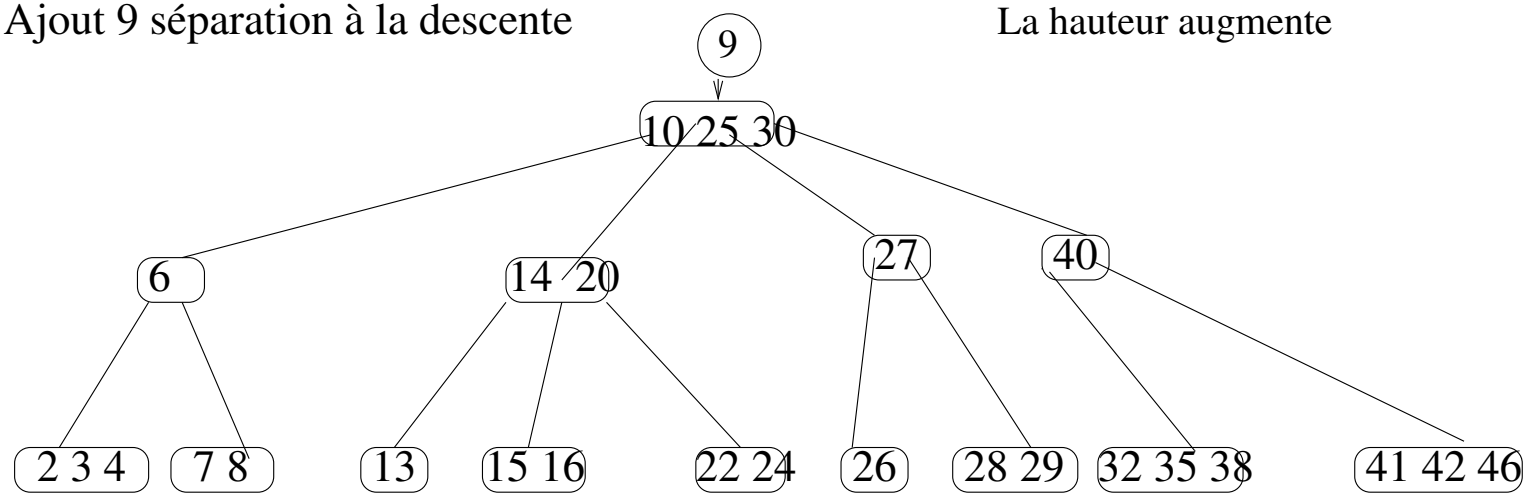
On sépare les nœuds à $2t - 1$ éléments rencontrés lors de la recherche de la position de x

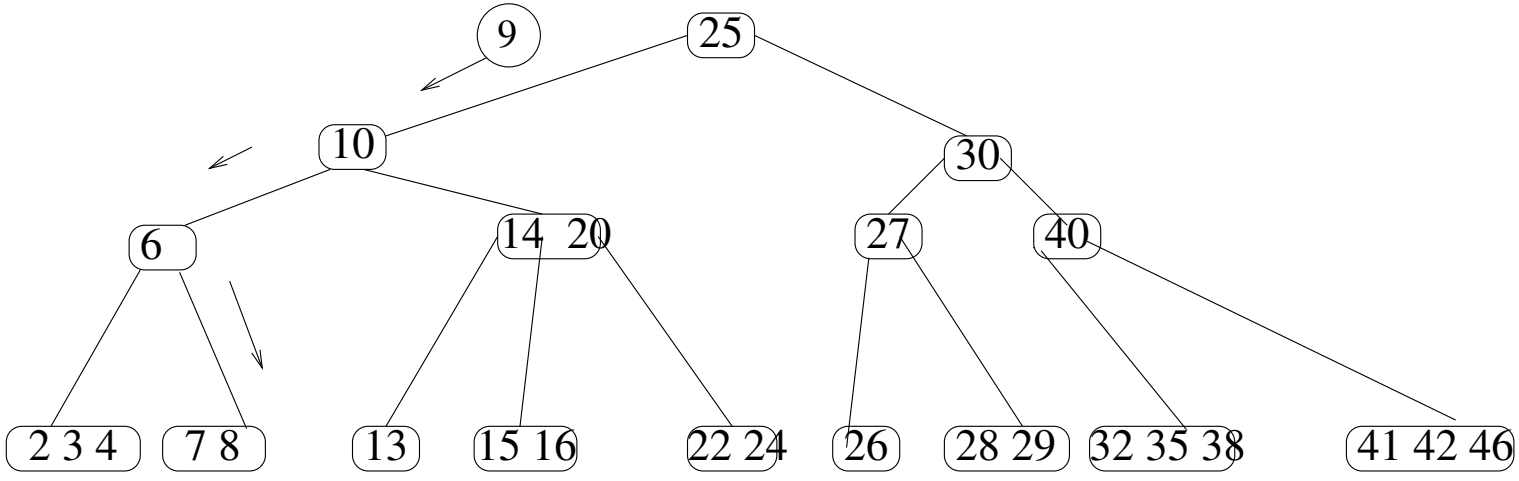
Ajout 46 séparation à la descente

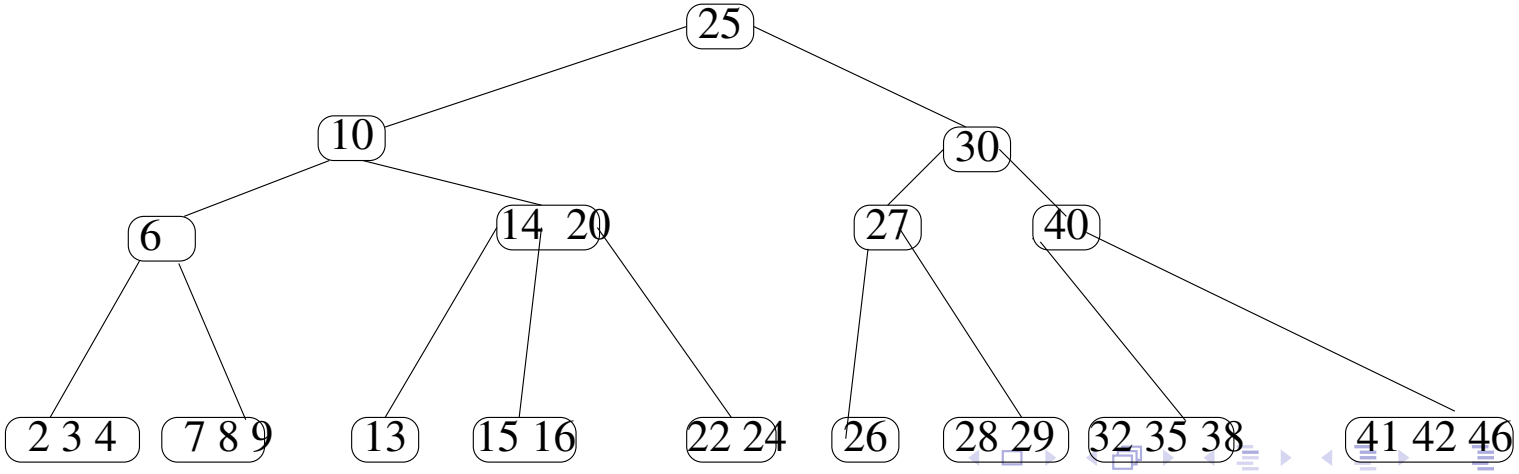


Ajout 9 séparation à la descente

La hauteur augmente







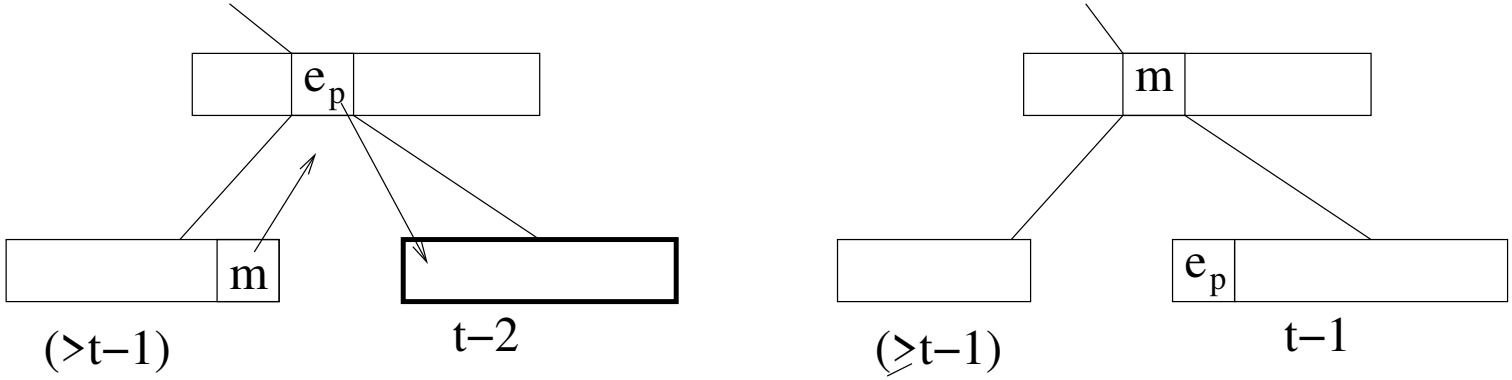
4 Suppression de l'élément x du Barbre A

```
Si A est vide
    retour
p = PositionPossible(x,A);
Si(p < 0 || x != e_p)  /* x peut etre dans A_{p+1} */
    Enlever  x de  A_{p+1}
Sinon  /* x == e_p */
    Si A est une feuille
        retirer x de A
    Sinon
        remplacer x par SupprimerMin de A_{p+1}
Equilibrer le nombre d'elements
```

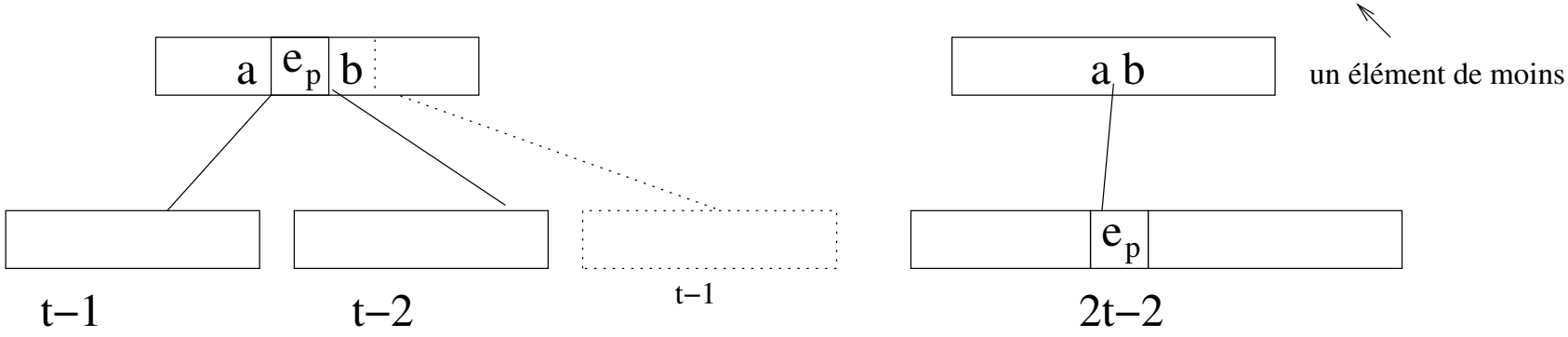
Equilibrage

On a supprimé un élément d'une feuille, soit directement soit par un appel à `SupprimeMin`.
Si cette feuille contient $t-2$ éléments, il faudra équilibrer ce qui peut entrainer un déséquilibre des ancêtres.
Quand un nœud contient $t-2$ éléments, il récupère un élément d'un de ses frères adjacents ou bien il est fusionné avec un de ses frères adjacents.

Si un frère adjacent a suffisament d'éléments, on récupère la valeur extrème du frère par rotation avec la valeur intermédiaire du nœud père.



Sinon on fusionne les deux enfants autour de l'élément e_p supprimé du nœud parent.



C'est ainsi que l'arbre diminue de hauteur. Avec cette méthode, le besoin de rééquilibrage peut remonter le long d'une branche.
Pour rester local on peut effectuer les traitements de fusion à la descente.

Fusion à la descente

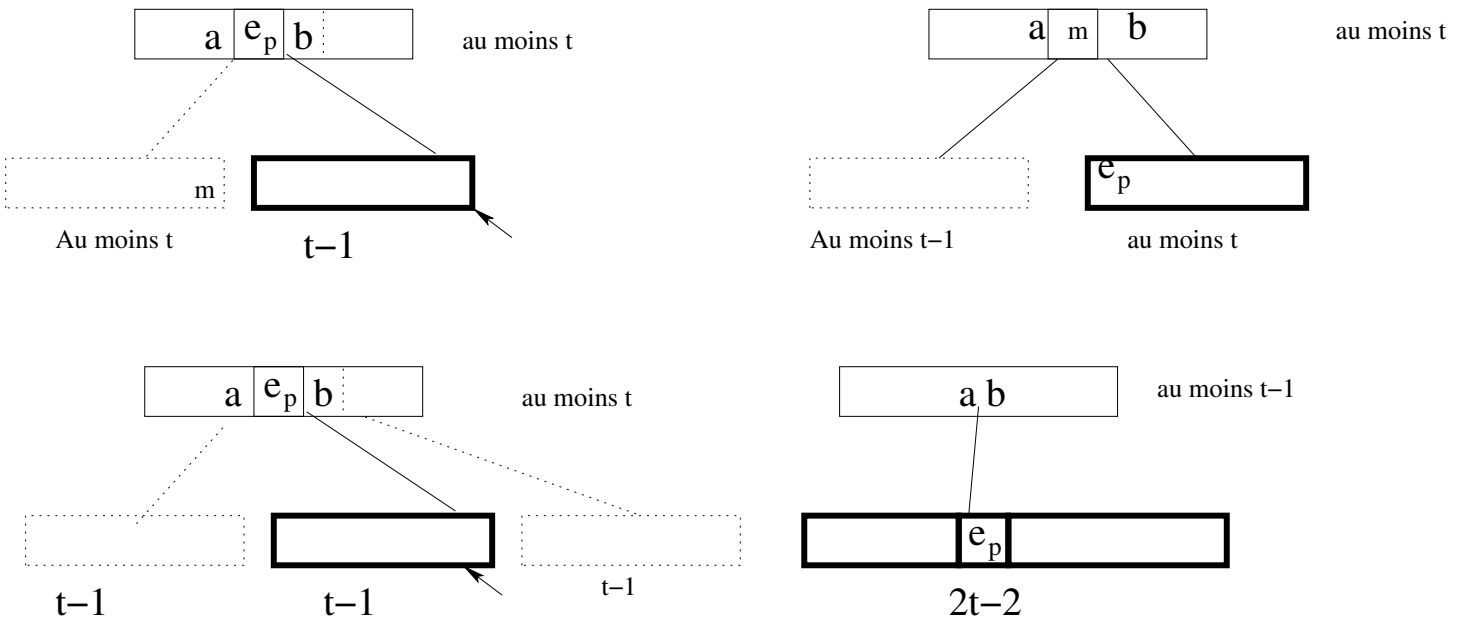
On s'assure que tous les nœuds traversés ont au moins t éléments (ce qui permettra si nécessaire d'en récupérer un).

Si le nœud à traverser possède $t - 1$ éléments,

on recupére un élément dans un frère adjacent si ce frère en a au moins t

on fusionne avec le frère si le frère a $t - 1$ éléments. On cherche x dans le nœoeud ou on descend.

FUSION A LA DESCENTE, X NON TROUVE



Si A est un nœud interne contenant x

Si l'un des fils A_p ou A_{p+1} contient plus de t éléments, on utilise `SupprimeMax`(A_p) ou `SupprimeMin`(A_{p+1})

`Si les deux fils ont` $t-1$ `éléments, on les fusionne (autour de x) avant de continuer la suppression dans le sous arbre créé.`

FUSION A LA DESCENTE, X TROUVE