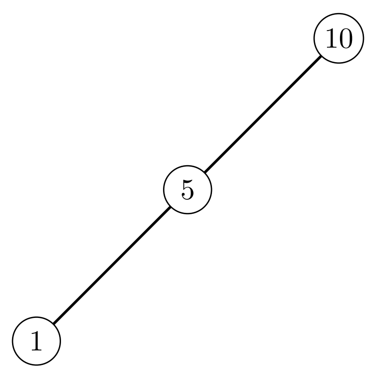
Corrections TD 1. Algorithmique Avancée : Arbres Binaires de Recherche Equilibrés

CERI - Licence 2 Informatique

Semestre 3

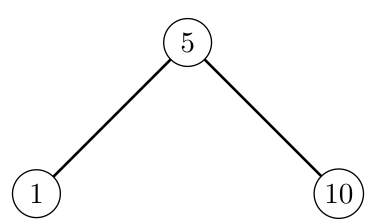
Exercice 2.

1/



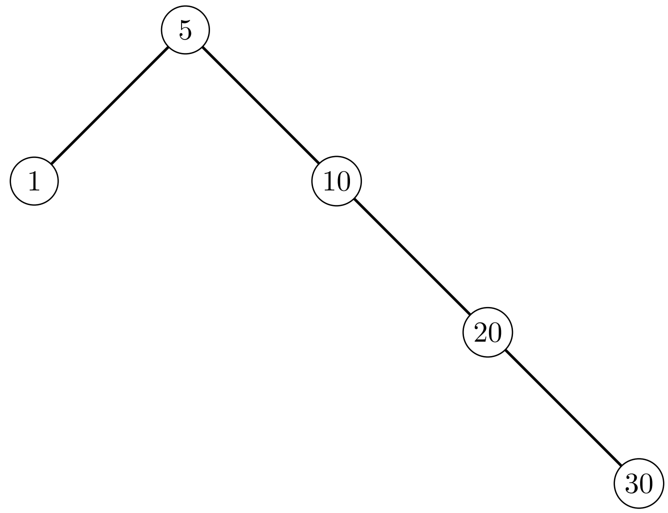
On note $d(v)$ le déséquilibre en tout noeud v. On a $d(10) = 2$, $d(5) = 1$, $d(1) = 0$.

2/



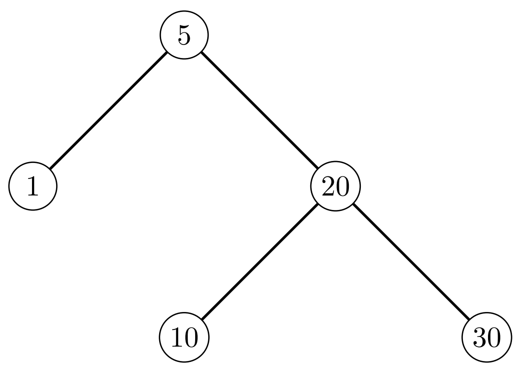
Après la rotation, les déséquilibres deviennent : $d(10) = 0$, $d(5) = 0$, $d(1) = 0$.

3/ On obtient après ajout de 20 et 30.



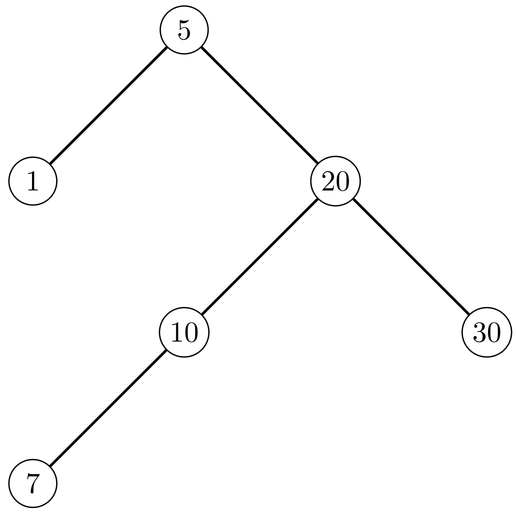
Les déséquilibres sont $d(5) = 2$, $d(1) = 0$, $d(10) = -2$, $d(20) = -1$, $d(30) = 0$.

4/ On ré-équilibre l'arbre par une rotation gauche en 10. Ce qui donne :



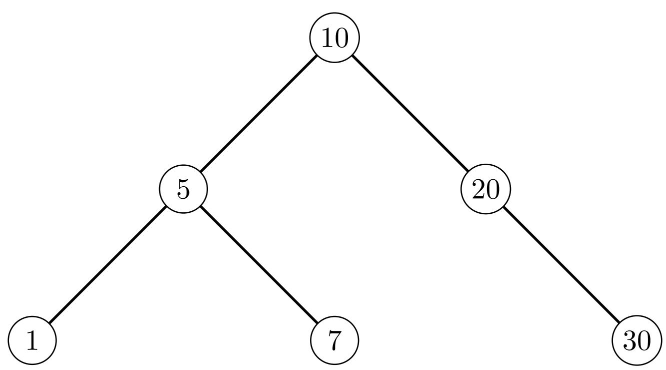
Après la rotation, les déséquilibres deviennent : $d(5) = -1$, $d(1) = 0$, $d(10) = 0$, $d(20) = 0$, $d(30) = 0$.

5/ L'ajout de 7 donne.



Les déséquilibres sont : $d(5) = -2$, $d(1) = 0$, $d(10) = 1$, $d(20) = 1$, $d(30) = 0$, $d(7) = 0$.

6/ Il faut appliquer une rotation droite-gauche. On obtient :



Avec les déséquilibres suivants : : $d(5) = 0$, $d(1) = 0$, $d(10) = 0$, $d(20) = -1$, $d(30) = 0$, $d(7) = 0$.

Exercice 4.

```
Algorithme : Test(r)
/* La méthode prend la racine r de l'arbre en argument */
/* minimum(r) est la méthode renvoyant l'adresse du noeud
   contenant la clé de valeur minimale */
/* successeur(v) est la méthode renvoyant l'adresse du noeud
   dont la clé est immédiatement supérieure à celle de dans
   l'arbre */
abr = vrai;
v = minimum(r);
tant que v ≠ null et abr = vrai faire
    w = successeur(v);
    si w ≠ null et w.cle ≤ v.cle alors
        abr = faux;
    fin
    v = w;
fin
retourne(abr)
```

Exercice 6.

1/ Dans ce cas il suffit de faire $r = y$.

2/ On insère y à la racine du sous-arbre gauche puis on effectue une rotation droite pour amener y en racine de l'arbre. Cela donne les deux instructions suivantes :

— $insertionRacine(r.fg, y)$
— $rotationDroite(r)$

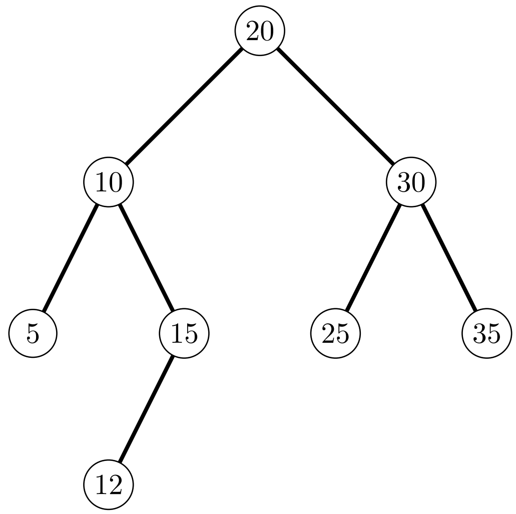
3/ De façon analogue à la question précédente, on insère y à la racine du sous-arbre droit puis on effectue une rotation gauche pour amener y en racine de l'arbre. Cela donne les deux instructions suivantes :

— $insertionRacine(r.fd, y)$
— $rotationGauche(r)$

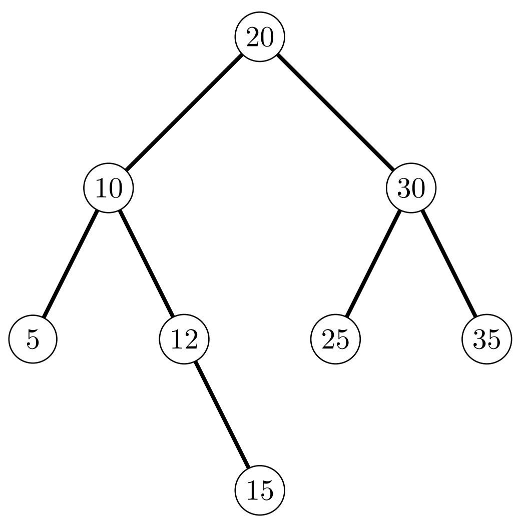
4/ En résumé on obtient l'algorithme ci-dessous.

```
1  Algorithme : insertionRacine(r, y)
2  si r == NULL alors
3  |   r = y;
4  fin
5  sinon
6  |   si y.cle < r.cle alors
7  |   |   si r.fg == NULL alors
8  |   |   |   r.fg = y
9  |   |   fin
10 |   |   sinon
11 |   |   |   insertionRacine(r.fg, y);
12 |   |   fin
13 |   |   rotationDroite(r);
14 |   fin
15 |   sinon
16 |   |   si r.fd == NULL alors
17 |   |   |   r.fd = y
18 |   |   fin
19 |   |   sinon
20 |   |   |   insertionRacine(r.fd, y);
21 |   |   fin
22 |   |   rotationGauche(r);
23 |   fin
24 fin
```

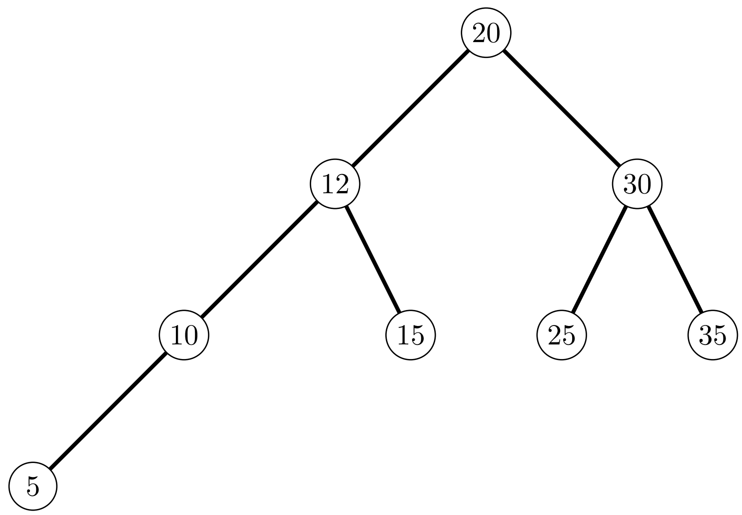
5/ Les appels récursifs s'arrêteront quand x pointera sur 15. A ce moment là, comme $12 < 15$, les lignes 7 et 8 donneront le positionnement initiale de 12 suivant :



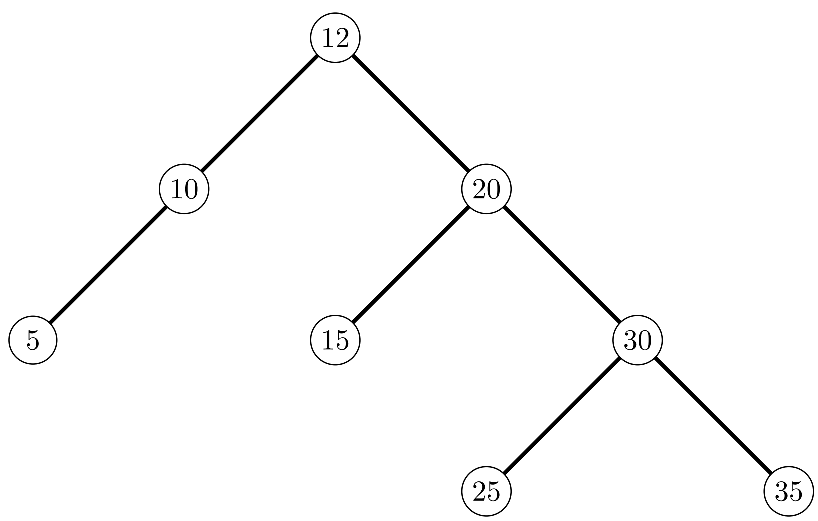
La remontée récursive effectuera alors une rotation à droite à partir de 15. Ce qui donnera



La remontée récursive effectuera ensuite une rotation à gauche à partir de 10. Ce qui donnera :



Et la remontée récursive effectuera ensuite une rotation à droite à partir de 20. Ce qui donnera finalement :



Exercice 7.

1/ Si $t = k - 1$ cela signifie, qu'à gauche de r, il y a $k - 1$ noeuds toutes de valeurs inférieures à $r.cle$. $r.cle$ contient donc la k-ième plus petite valeur. Il suffit donc de retourner r qui est déjà en racine.

2/ Si $t > k - 1$, le sous-arbre gauche contient plus de $k - 1$ noeuds. La k-ième plus petite valeur est donc dans ce sous-arbre. Il faut relancer récursivement la méthode dans celui pour le trouver. Par hypothèse de récurrence, le noeud contenant la k-ième plus petite valeur se retrouvera en racine du sous-arbre gauche. Ca sera précisement $r.fg$. La dernière étape consistera à le remonter en racine par une rotation droite. On a donc les deux instructions ci-desosus à réaliser :

— $partition(r.fg, k)$
— $rotationDroite(r)$

3/ Par un raisonnement analogue, le sous-arbre gauche contient strictement moins de $k-1$ noeuds. La k-ième plus petite valeur se trouvera donc dans le sous-arbre droit de r. Elle correspond à la $(k-t-1)$-ième plus petite valeur du sous-arbre droit. Par exemple, si on avait $k = 4$, et $t = 0$ (i.e le fils gauche est vide), le 4-ième plus petit élement correspond au 3-ième plus petit du sous-arbre droit. On partitionne donc le sous-arbre droit par rapport à cette $(k-t-1)$-ième plus petite valeur, que l'on fait remonter ensuite en racine avec une rotation gauche.
— $partition(r.fd, k-t-1)$
— $rotationGauche(r)$

4/ L'algorithme récursif de partitionnement s'en déduit.

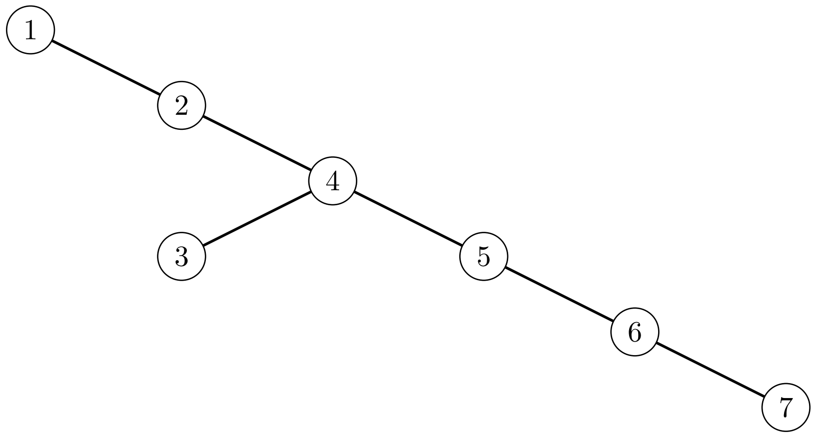
```
Algorithme : partition(r, k)
res = NULL;
si r ≠ NULL alors
    t = 0;
    si r− > fg alors
        t = r− > fg− > N
    fin
    si t == k − 1 alors
        res = x;
    fin
    sinon
        si t > k − 1 alors
            res = partition(r− > fg, k);
            rotationDroite(r)
        fin
        sinon
            res = partition(r− > fd, k − t − 1);
            rotationGauche(r)
        fin
    fin
fin
retourner(res);
```

Exercice 8.

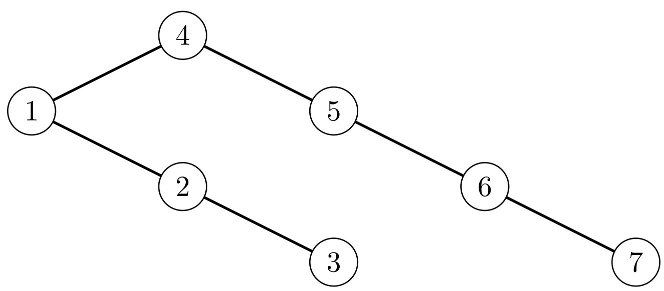
A la première itération on a $m = 4$. On partitionne par rapport à la 4-ième plus petite valeur avec l'instructions $partition(r- > fg, 4)$. La suite d'appels de la méthode partition est la suivante.

- Dans $partition(r, 4)$, on a $t = 0$, ce qui générera l'appel récursif suivant
- $partition(r- > fd, 3)$. Comme $t = 0$ de nouveau, l'autre appel ci-dessous s'effectue.
- $partition(r- > fd, 2)$. $t = 0$ de nouveau, et déclenche le dernier appel,
- $partition(r- > fd, 1)$.

La remontée récursive active une rotation gauche au noeud 3. Ce qui donne :



Puis une rotation gauche en 3, suivie d'une autre en 1. Ce qui donne finalement :



"equilibre" est alors exécutée récursivement sur les fils gauche et droit de 4. On obtient après ces exécutions :