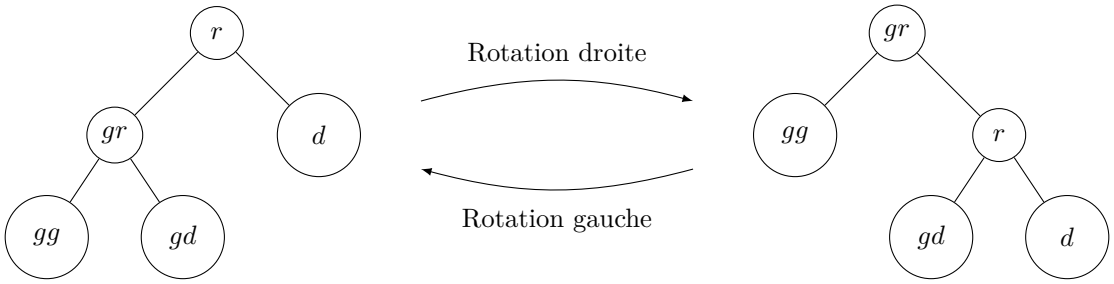
I Arbre AVL

Un arbre AVL est un ABR tel que, pour chaque noeud, la différence de hauteur de ses sous-arbres gauche et droit soit au plus 1. Pour éviter de calculer plusieurs fois la même hauteur, on stocke, dans chaque noeud, la hauteur de l'arbre correspondant. On utilise donc le type suivant: `type 'a avl = V | N of 'a * 'a avl * 'a avl * int`.

1. Montrer qu'un arbre AVL à n noeuds est de hauteur $O(\log(n))$. Soit $h \geq 1$ et `N(r, g, d)` un AVL de hauteur h ayant u_h noeuds. Alors `g` ou `d` est de hauteur $h-1$ et l'autre de hauteur $\geq h-2$, donc ont au moins u_{h-1} et u_{h-2} sommets, i.e $u_h \geq u_{h-1} + u_{h-2} + 1 \geq u_{h-1} + u_{h-2}$. Comme de plus $u_0 = 0 = f_0$, $u_h \geq f_h$, où f_h est la suite de Fibonacci définie par $f_n = f_{n-1} + f_{n-2}$ (on peut montrer $u_h \geq f_h$ par récurrence). Comme on sait que (cf équation récurrente d'ordre 2 du cours de maths) $f_n = \Theta(\phi^n)$ avec $\phi = \dfrac{1+\sqrt{5}}{2}$ (nombre d'or), $u_h \geq K\phi^h$ puis $h \leq \log_\phi u_h - K'$ pour des constantes K, K' et h assez grand.
   Donc la hauteur h d'un AVL à n sommets vérifie $h \leq \log_\phi u_h \leq \log_\phi n \approx 1.44 \log_2(n)$.
2. Écrire une fonction utilitaire `ht : 'a avl -> int` donnant la hauteur d'un AVL.
3. Écrire une fonction utilitaire `node : 'a -> 'a avl -> 'a avl -> 'a avl` construisant un AVL à partir d'une racine et de ses deux sous-arbres.

Lors de l'ajout d'un élément (en tant que feuille) la condition d'AVL peut être violée et doit être rétablie. Pour cela, on introduit l'opération de *rotation droite* (et son symétrique *rotation gauche*) autour d'un noeud:



4. Est-ce qu'une rotation préserve la propriété d'ABR?
5. Écrire une fonction `rotd` réalisant une rotation droite sur un arbre supposé de forme convenable.
   On suppose dans la suite avoir aussi une fonction `rotg` réalisant une rotation gauche (opération inverse).

On suppose que, après l'ajout d'un élément, `g` et `d` sont des AVL, mais que `N(r, g, d)` n'est pas un AVL. Pour les deux questions suivantes, on suppose que `ht g > ht d + 1`, l'autre cas étant symétrique. On décompose `g` en `N(gr, gg, gd)`.

6. Si `ht gg > ht gd`, montrer qu'une rotation suffit pour transformer `N(r, g, d)` en AVL.
7. Sinon, `ht gg < ht gd`. Montrer comment se ramener au cas précédent en une rotation.
8. En déduire une fonction `balance` prenant `r`, `g`, `d` en arguments et renvoyant l'AVL correspondant.
9. En déduire une fonction `add` ajoutant un élément dans un AVL en conservant la structure d'AVL.
10. Écrire aussi des fonctions `del` et `has` pour supprimer un élément et savoir si un élément appartient à un AVL. On supposera qu'il n'y a pas de doublon dans l'AVL. Complexité de ces fonctions?

II Implémentation de dictionnaire avec arbre

Écrire des fonctions `dict_get : 'a -> ('a * 'b) avl -> 'b option`,
`dict_add : 'a * 'b -> ('a * 'b) avl -> ('a * 'b) avl` `dico_del`, qui permettent d'implémenter un dictionnaire à l'aide d'un AVL, en mettant un couple (clé, valeur) sur chaque noeud (`'a` étant le type des clés et `'b` celui des valeurs). Complexités ?
Remarque : On pourrait faire de même avec un ARN ou n'importe quel arbre binaire de recherche.

III Problème de géométrie

On considère un ensemble E de n points dans le plan $\mathbb{R}^2$ (représenté par un tableau de couples, par exemple). Donner un algorithme pour trouver le nombre de rectangles que l'on peut former en choisissant 4 points de E. On pourra d'abord donner une solution simple en $O(n^4)$ puis une solution plus efficace, en utilisant un dictionnaire (quelle complexité obtient-on alors avec un dictionnaire implémenté par table de hachage? par arbre binaire de recherche équilibré ?).