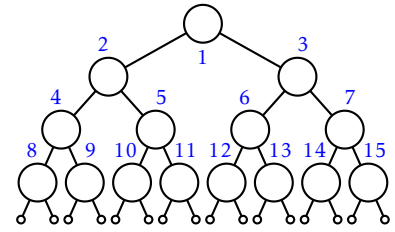


Lorsque l'arbre est équilibré au maximum, alors il contient entre 2^{h-1} et $2^h - 1$ nœuds. Or, si $2^{h-1} \leq k < 2^h$, alors $\lfloor \log_2(k) \rfloor = h - 1$. Si l'arbre est de taille N, on a donc :

$$\lfloor \log_2(N) \rfloor + 1 \leq h \leq N$$

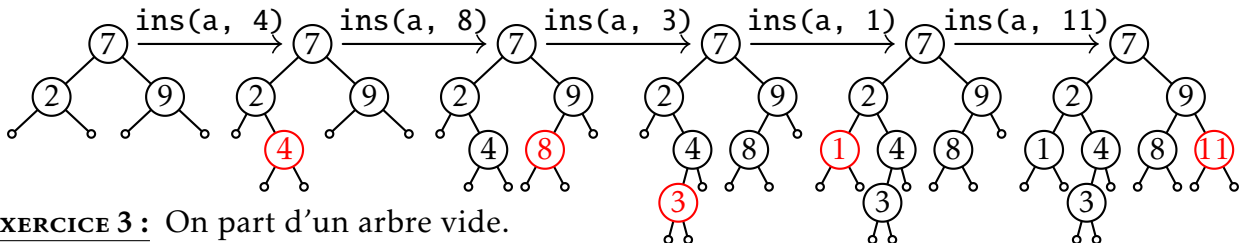


Dans le meilleur des cas, on retrouve une complexité logarithmique, comme pour une recherche dichotomique.

Ajout d'un élément

Si on veut qu'un ABR soit efficace pour la recherche, il faut faire attention à la façon dont il est construit. Tout d'abord, il faut voir comment insérer les valeurs. On peut utiliser l'algorithme ci-contre. On suppose que l'arbre est modifié par la fonction et que `creer_feuille` transforme l'arbre vide en un arbre ne contenant qu'un nœud.

```
def inserer(arbre, val):
    if est_vide(arbre):
        creer_feuille(arbre, val)
    elif val < racine(arbre):
        inserer(gauche(arbre), val)
    else:
        inserer(droite(arbre), val)
```



EXERCICE 3 : On part d'un arbre vide.

- 1) Dessiner l'ABR obtenu, en partant de l'arbre vide, après l'insertion, dans l'ordre, de 9, 4, 6, 10, 3, 5, 15, 2, 8 et 11
- 2) Rajouter 9 et 4 à l'arbre.

On peut remarquer que notre fonction n'empêche pas de rajouter plusieurs fois la même valeur dans l'arbre. Selon les besoins, on pourra interdire un tel ajout, ou l'autoriser.

Au niveau de la complexité, lors d'un ajout, on fait le même type de parcours que pour une recherche. La complexité est donc logarithmique dans le meilleur des cas et linéaire dans le pire. Néanmoins, si les valeurs sont ajoutées "au hasard", la complexité de chaque ajout reste logarithmique. Il faut surtout éviter de trier les valeurs avant de les insérer.

La suppression est plus complexe puisqu'elle nécessite, dans certains cas, de modifier la structure de l'arbre autour du nœud modifié. La complexité reste néanmoins la même que pour l'ajout ou la recherche.

Comparaison avec les autres structures

Nous avons vu, qu'en moyenne, l'ajout et la recherche d'éléments dans un ABR est logarithmique. Pour un tableau, la recherche est logarithmique, grâce à la dichotomie, mais on ne peut pas ajouter de valeurs, à moins de devoir tout recopier dans un nouveau tableau, ce qui est linéaire. Si toutes les valeurs sont connues à l'avance, si on doit les trier, la complexité devient $O(n \log n)$. Avec une liste chaînée, l'insertion est plus rapide, tout en restant linéaire et la recherche est également linéaire puisqu'on ne peut pas accéder directement à une valeur par son indice.

Globalement, les tableaux sont intéressants si les données sont statiques, mais si on doit en ajouter, les ABR sont plus efficaces. Afin de garantir que la complexité reste logarithmique, il est possible d'utiliser des arbres AVL ou "rouge-noir", pour lesquels l'équilibrage est garanti à l'insertion ou à la suppression des valeurs. Ces opérations restent logarithmiques.