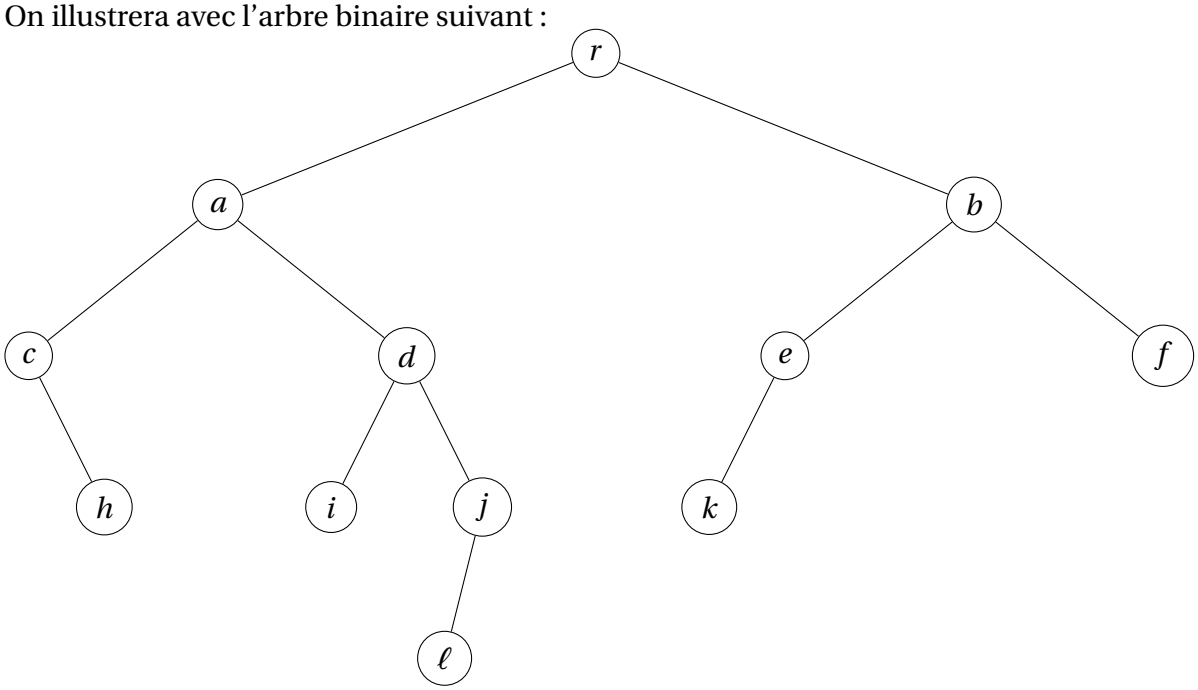
Parcours d'un arbre binaire

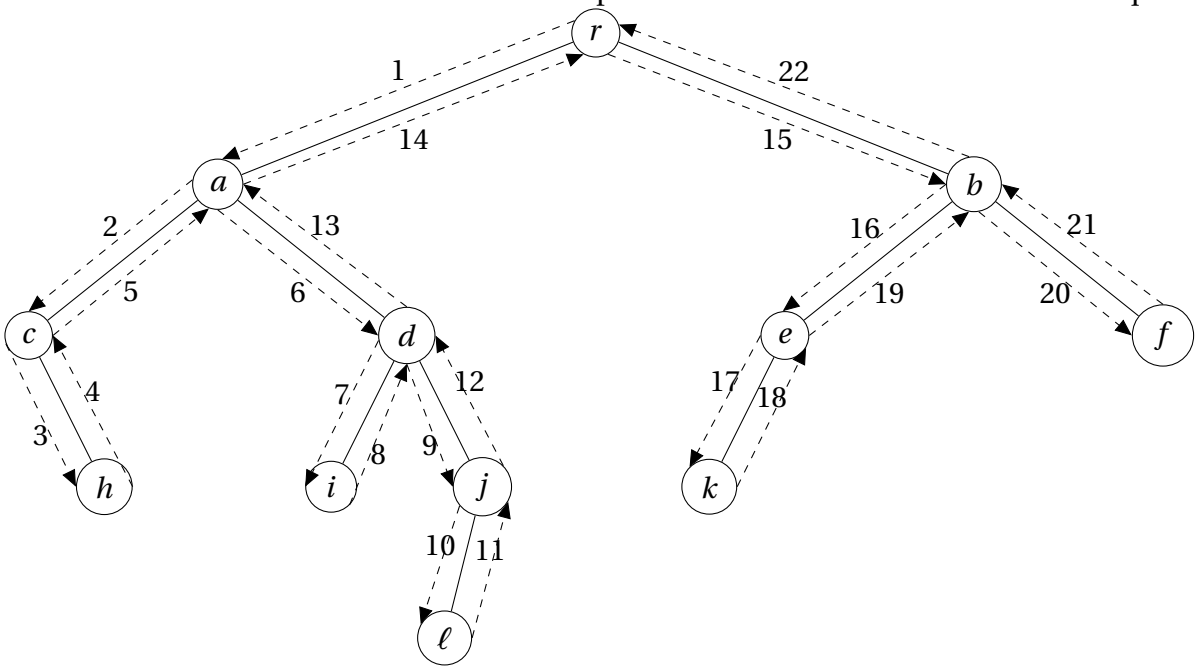
Un arbre binaire est un arbre avec racine dans lequel tout noeud a au plus deux fils : un éventuel fils gauche et un éventuel fils droit.

On illustrera avec l'arbre binaire suivant :



1 Balade autour de l'arbre

On se balade autour de l'arbre en suivant les pointillés dans l'ordre des numéros indiqués :



1.1 Première définition des trois parcours

A partir de ce contour, on définit trois parcours des sommets de l'arbre :

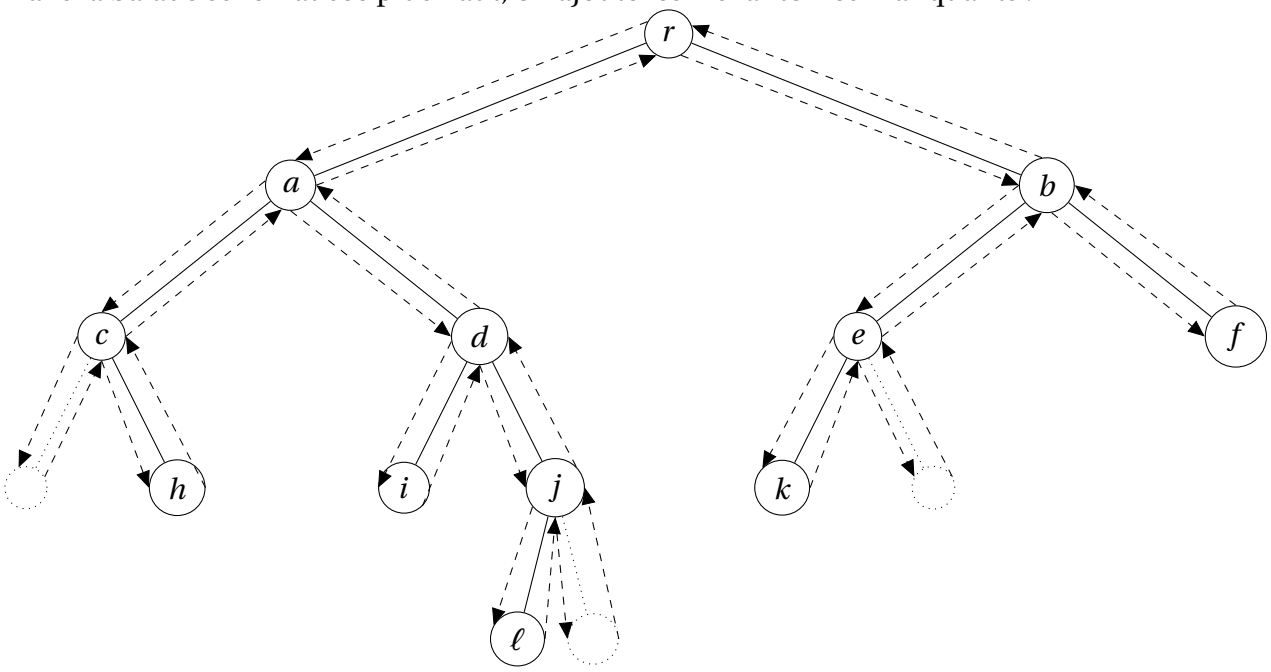
1. l'ordre préfixe : on liste chaque sommet la première fois qu'on le rencontre dans la balade. Ce qui donne ici : ...
2. l'ordre postfixe : on liste chaque sommet la dernière fois qu'on le rencontre. Ce qui donne ici : ...
3. l'ordre infixe : on liste chaque sommet ayant un fils gauche la seconde fois qu'on le voit et chaque sommet sans fils gauche la première fois qu'on le voit. Ce qui donne ici : ...

Une résolution

1. ordre préfixe : $r, a, c, h, d, i, j, \ell, b, e, k, f$.
2. ordre postfixe : $h, c, i, \ell, j, d, a, k, e, f, b, r$.
3. ordre infixe : $c, h, a, i, d, \ell, j, r, k, e, b, f$.

1.2 Seconde définition des trois parcours

Dans la balade schématisée plus haut, on ajoute les fils fantômes manquants :



On peut ainsi considérer qu'on passe une fois à gauche de chaque noeud (en descendant), une fois en-dessous de chaque noeud, une fois à droite de chaque noeud (en remontant).
Vérifier, sur l'exemple, que chacun des ordres préfixe, infixe, postfixe est obtenu en listant tous les mots :
– soit lorsqu'on passe à leur gauche,
– soit lorsqu'on passe à leur droite,
– soit lorsqu'on passe en-dessous.

Une résolution

1. A gauche : préfixe.

2. A droite : postfixe.
3. En-dessous : infixe.

2 Algorithmes récursifs

Pour chacun des parcours définis ci-dessus (postfixe, infixe, préfixe), définir récursivement le parcours.

Une résolution

1. Parcours préfixe.

Pseudo-code

```
ParcoursPréfixe(Arbre binaire T de racine r)
        Afficher  clef[r]
        ParcoursPréfixe(Arbre de racine fils_gauche[r])
        ParcoursPréfixe(Arbre de racine fils_droit[r])
```

2. Parcours postfixe.

Pseudo-code

```
ParcoursPostfixe(Arbre binaire T de racine r)
        ParcoursPostfixe(Arbre de racine fils_gauche[r])
        ParcoursPostfixe(Arbre de racine fils_droit[r])
        Afficher  clef[r]
```

3. Parcours infixe.

Pseudo-code

```
ParcoursInfixe(Arbre binaire T de racine r)
        ParcoursInfixe(Arbre de racine fils_gauche[r])
        Afficher  clef[r]
        ParcoursInfixe(Arbre de racine fils_droit[r])
```

3 Représentation en machine

Chaque nœud de l'arbre T est representé par un objet ayant un champ clef (des valeurs à trier par exemple), un champ père, un champ fils_gauche, un champ fils_droit (stockent des pointeurs).
Lorsque père[x]=NIL, x est la racine de l'arbre. Lorsque x n'a pas de fils gauche, fils_gauche[x]=NIL (idem pour fils_droit). La racine de l'arbre T est pointée par l'attribut racine[T]. Lorsque racine[T]=NIL, l'arbre est vide.

4 Complexité d'un parcours infixe

Vérifier qu'avec n noeuds, le parcours infixe

Pseudo-code

```
Parcours_Infixe(arbre binaire T de racine x)
        Si x distinct de NIL *****temps constant T(0)=c pour un sous-arbre vide
        alors
        Parcours_Infixe( arbre de racine fils_gauche[x]) *****temps T(k)
        Afficher clef[x]  *****temps constant d
        Parcours_Infixe( arbre de racine fils_droit[x]) *****temps T(n-k-1)
        FinSi
```

prend un temps en $\Theta(n)$ (établir avec les notations suggérées ci-dessus : $T(n) = (c+d)n+c$)

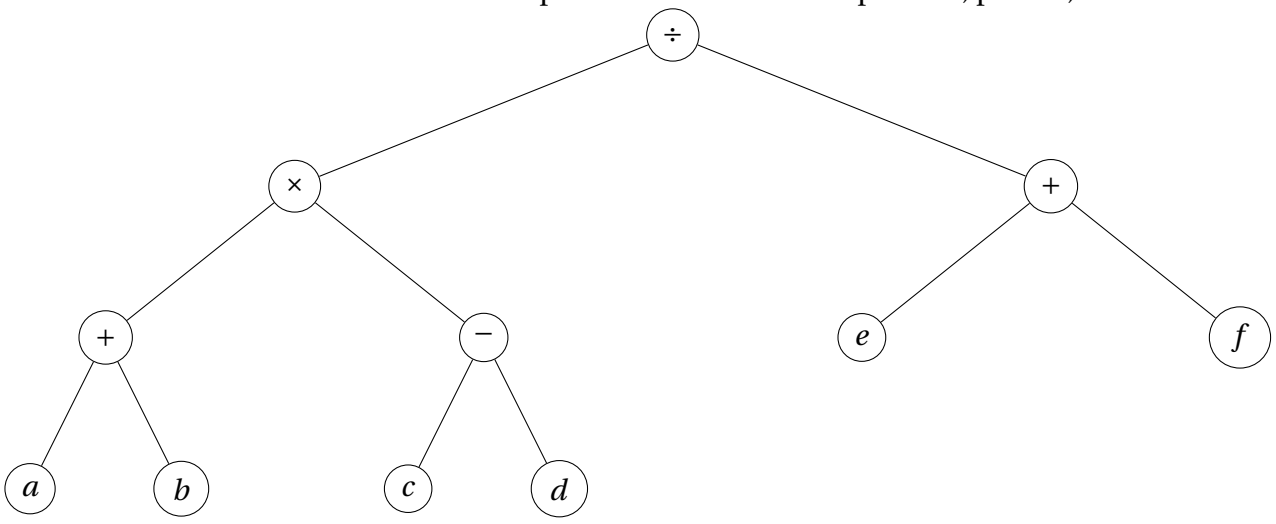
Une résolution

Amorce : $T(0) = c = (c+d)\times 0 + c$.
Hérédité :

$$\begin{aligned} T(n) &= T(k) + T(n-k-1) + d \\ &= [(c+d)k+c] + [(c+d)(n-k-1)+c] + d \\ &= (c+d)n + c - (c+d) + c + d \\ &= (c+d)n + c \end{aligned}$$

5 La notation polonaise inverse

Écrire les sommets de l'arbre ci-dessous pour chacun des ordres postfixe, préfixe, infixe :



Pour le parcours infixe, on ajoute la convention suivante : on ajoute une parenthèse ouvrante à chaque fois qu'on entre dans un sous-arbre et on ajoute une parenthèse fermante lorsqu'on quitte ce sous-arbre.

Une résolution

1. Préfixe (notation polonaise) : $\div, \times, +, a, b, -, c, d, +, e, f$.
2. Postfixe (polonaise inverse) : $a, b, +, c, d, -, \times, e, f, +, \div$.
3. Infixe : $a, +, b, \times, c, -, d, \div, e, +, f$.
   Avec un ajout de parenthèses (ouvrante en rencontrant le nœud racine du sous arbre pour la première fois et fermante lorsqu'on le rencontre pour la dernière fois, avec exception sur les sous-arbres constitués d'une feuille) : $\big((a+b)\times(c-d)\big)\div(e+f)$.

L'infixe nécessite cette convention pour lever les ambiguïtés, les deux autres non. La préfixe consiste à voir les opérateurs comme des fonctions de deux variables :

$$\div, \times, +, a, b, -, c, d, +, e, f = \div\Big(\times\left[+(a,b), -(c,d)\right], +(e,f)\Big)$$

Idem avec la postfixe mais avec la fonction écrite sur la droite.

6 Le tri du bijoutier

On dispose d'une liste de nombres. Par exemple, la liste 7, 9, 3, 5, 4, 1, 8. On associe à chaque élément n de la liste un nœud v (initialisation : père[v]=NIL, fils_gauche[v]=NIL, fils_droit[v]=NIL, clef[v]=n).

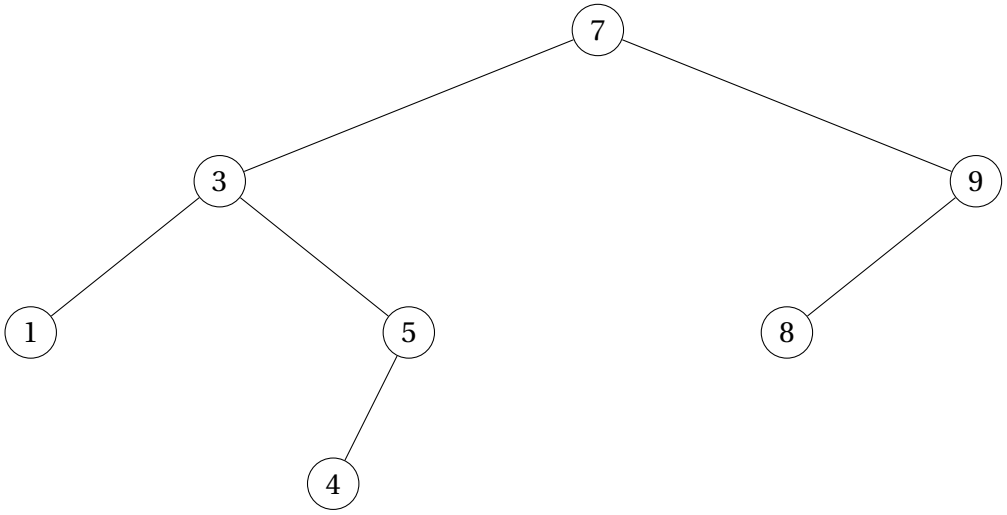
Pseudo-code

```
Arbre_Insérer(Arbre T, noeud z)
        y:=NIL
        x:=racine[T]

        TantQue x distinct de NIL faire
                y:=x
                Si clef[z]<clef[x]
                alors x:=fils_gauche[x]
                sinon x:=fils_droit[x]
                FinSi
        FinTantQue

        père[z]:=y
        Si y=NIL
        alors racine[T]:=z
        sinon
                Si clef[z]<clef[y]
                alors fils_gauche[y]:=z
                sinon fils_droit[y]:=z
                FinSi
         FinSi
```

1. Dresser l'arbre obtenu en appliquant l'algorithme Arbre_Insérer aux éléments de la liste (dans l'ordre de la liste) en partant d'un arbre vide pour le premier élément, chaque appel à l'algorithme modifiant l'arbre.
2. L'un des parcours postfixe, infixe, préfixe de la liste trie la liste. Lequel ?
3. Dans la construction de l'arbre pour une liste de n nombres, quel est le nombre de comparaisons effectuées dans le pire des cas ?
4. Quel est le nombre de comparaisons effectuées si l'arbre final est un arbre binaire complet (arbre binaire dans lequel tout nœud autre qu'une feuille a deux fils et dans lequel les feuilles sont tous des nœuds de même profondeur).

Une résolution

1. L'arbre obtenu :



2. Le parcours infixe trie la liste. Les éléments de gauche sont en effet par construction plus petits qu'un noeud et sont affichés avant le nœud dans l'ordre infixe et les éléments de droite qui sont, par construction, plus grands sont affichés dans l'ordre infixe après le nœud. Par "récurrence", on a donc un affichage des éléments de la liste dans l'ordre.

   On peut donner une version graphique de ce parcours en projetant verticalement les nœuds sur une droite horizontale (à dessiner sous l'arbre).
3. Le pire des cas correspond aux cas où la liste est triée (ordre croissant ou décroissant). Le nombre de comparaisons à effectuer est alors de $1+2+\cdots+(n-1)=\frac{1}{2}n(n-1)$.
4. Pour ajouter un noeud au niveau de profondeur p (la racine étant au niveau de profondeur 0), on effectue p comparaisons. Si la profondeur est h, on aura effectué une comparaison pour chacun des deux noeuds de profondeur 1 (2×1), deux comparaisons pour chacun des 2^2 noeuds de profondeur 2 (total $2^1+2\times 2^2$), tros comparaisons pour chacun des 2^3 noeuds de profondeur 3 (total $2^1+2\times 2^2+3\times 2^3$) ...

   Le nombre de comparaisons pour une profondeur h est (preuve facile par récurrence) :

$$\sum_{j=1}^{h} j\times 2^j=(h-1)\times 2^{h+1}+2$$

Avec n noeuds (c'est à dire n nombres à trier), on a $1+2+2^2+\cdots+2^h = 2^{h+1}-1$ et

$$\sum_{j=1}^{h} j \times 2^j = (h-1)\times 2^{h+1}+2 = (\log_2(n+1)-1)\times(n+1)+2$$

On a donc un cas optimal en $O(n\log(n))$ et on peut montrer (comme pour le quick sort) que la hauteur moyenne d'un arbre binaire de recherche construit aléatoirement à partir de n clefs est $O(\log(n))$ (référence : introduction à l'algorithmique, Cormen, Leiserson,Rivest, Stein, éditions Dunod, 2002, page 258, paragraphe 12.4).

Les caractéristiques de temps sont les mêmes que pour le quick sort, mais avec un avantage du côté des caractéristiques d'espace pour le quick sort (on trie le tableau sur place pour le quick sort, on crée un arbre de recherche pour le tri du bijoutier).

7 Références

1. Introduction à l'algorithmique. Auteurs : Cormen, Leiserson, Rivest, Stein. Edition française : Dunod 2002.

   Plus de 1100 pages sur les algorithmes et les structures de référence. Le chapitre 12 concerne les arbres binaires de recherche.

2. Un article de Jean-Claude Oriol sur le site de l'apmep avec un passage sur le tri du bijoutier :

   `http://www.apmep.asso.fr/spip.php?article3405`

3. Le cours de Pierre Audibert (Paris 8) en ligne :

   `http://www.ai.univ-paris8.fr/~audibert/ens/06-ARBREBINX.pdf`