Algorithmique Avancée

TD 3-4 : Arbres bicolores et tries

Eléments de solutions

1 Arbres bicolores

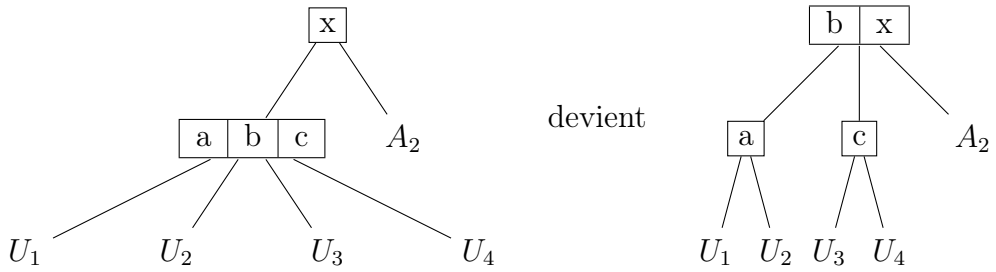
Solution

Conversion Les arbres bicolores permettent de représenter les arbres 2-3-4 avec des arbres binaires raisonnablement équilibrés et la conversion d'un type d'arbre à l'autre est aisée, comme on va le voir dans les exercices qui suivent.

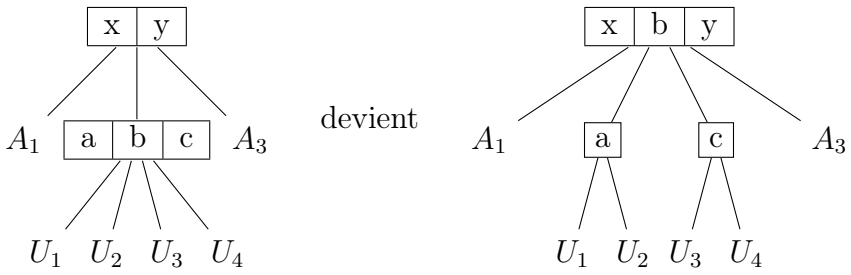
Exercice 1.1 ***: Arbres 2-3-4***

Solution

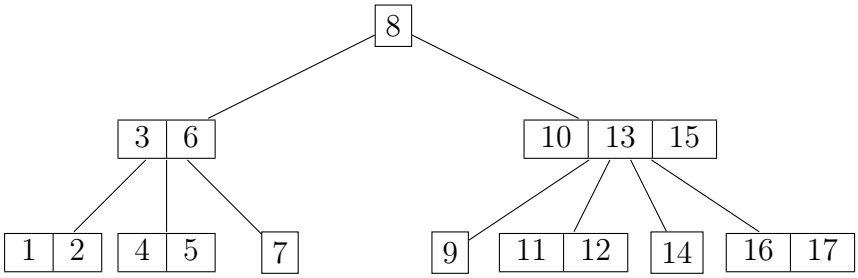
1. Du fait que les éclatements se font à la descente, le père d'un 4-noeud ne peut pas être un 4-noeud.
— exemple où père du 4-nœud est un 2-nœud



A faire avec le 4-noeud.à droite
— un exemple parmi 3 où le père du 4-nœud est un 3-nœud



2. Voilà l'arbre construit :

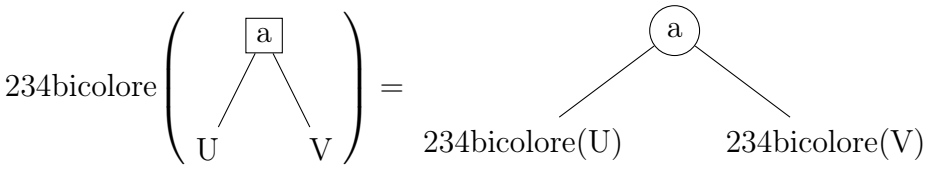


Exercice* 1.2 *: Transformation d'un arbre 2-3-4 en arbre bicolore

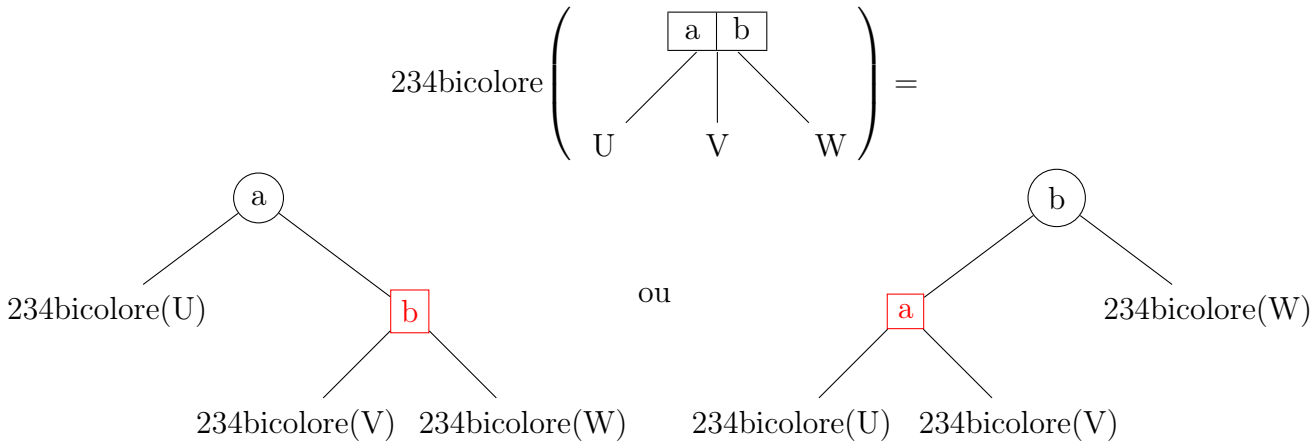
Solution

1. Dans mes dessins, les nœuds blancs seront ronds et noirs, et les nœuds rouges seront carrés.

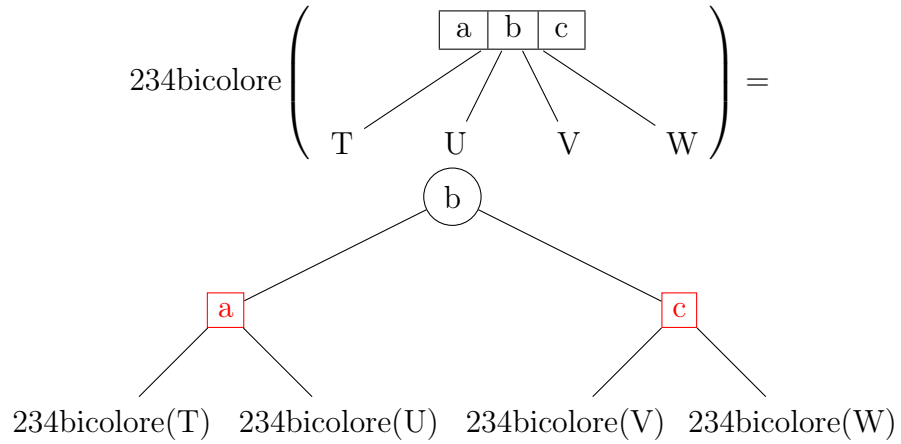
Règle 2 :



Règle 3 :



Règle 4 :



2. (a) Racine blanche par construction dans tous les cas

(b) Les feuilles sont introduites par l'algorithme au niveau des arbres vides sous la forme de feuilles blanches sans clef donc toutes les feuilles de l'arbre transformé sont blanches et sans clef

(c) Par construction, on introduit un sommet rouge soit comme le fils rouge d'un père blanc dans le cas d'un 3-noeud, soit comme l'un des deux fils rouges d'un père blanc dans le cas d'un 4-nœud, donc tous les sommets rouges ont un père blanc.

(d) Par récurrence sur la hauteur h de l'arbre 2-3-4 on montre à la fois c) et que dans tout chemin de la racine à une feuille on rencontre exactement $h+2$ sommets blancs (en comptant les extrémités)

C'est vrai pour un arbre de hauteur -1 : arbre vide devient une feuille blanche et le seul chemin possible contient 1 sommet blanc.

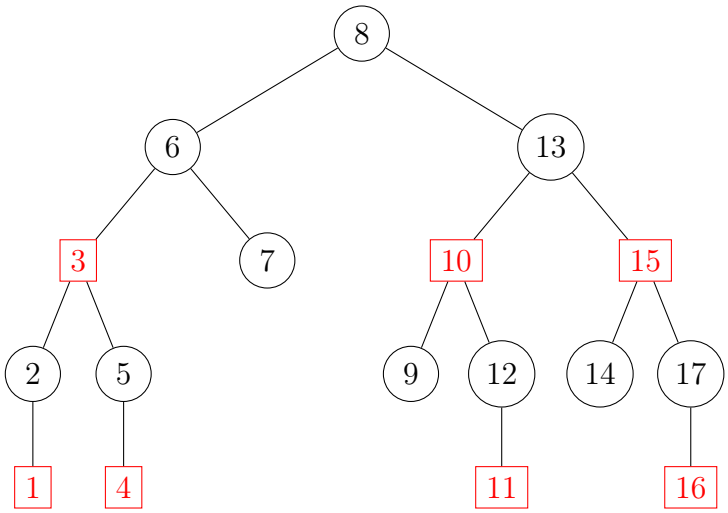
Soit A de hauteur h. Les sous-arbres de A sont des arbres 2-3-4 de hauteur $h-1$ (notés T, U ,V ou W dans le 1.)

Soit N un nœud de 234bicolore(A). Si N est dans un sous-arbre obtenu par un appel récursif

234bicolore sur T, U ,V ou W alors la propriété c résulte directement de l'hypothèse d'induction. Sinon N est un des nœuds a, b ou c du 1). Un chemin jusqu'à une feuille qui part d'un de ces nœuds passe par la racine N' de 234bicolore de T,U,V ou W. Par induction de N' à la feuille il contient $h+1$ sommets blancs. S'il part d'un sommet rouge alors le chemin de N à la feuille contient aussi $h+1$ sommets blancs. Et s'il part de la racine de 234bicolore(A) il en contient $h+1$.

On en déduit que la hauteur de 234bicolore(A) est comprise entre $h+1$ et $2(h+1)$.

3.



4.

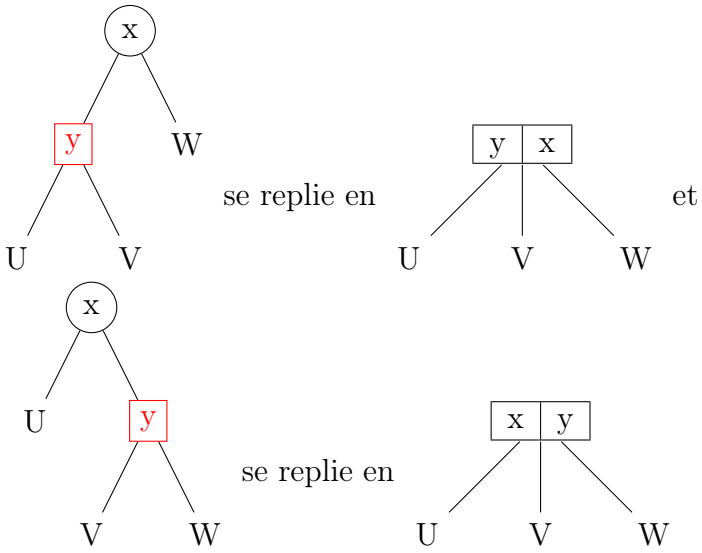
```
234bicolore (A)
si EstVide(A) alors retourne (FeuilleBlanche())
si Degre(A)=2 alors
  retourne (Arbre (Elem-1(A), blanc, 234bicolore(SSab-1(A)), 234bicolore(SSab-2(A))))
si Degre(A)=3 alors
  retourne (Arbre (Elem-2(A), blanc,
            Arbre (Elem-1(A), rouge, 234bicolore(SSab-1(A)), 234bicolore(SSab-2(A))))
            234bicolore(SSab-3(A)))
si Degre(A)=4 alors
  retourne (Arbre (Elem-2 (A), blanc,
            Arbre (Elem-1(A), rouge, 234bicolore(SSab-1(A)), 234bicolore(SSab-2(A))))
            Arbre (Elem-3(A), rouge, 234bicolore(SSab-3(A)), 234bicolore(SSab-4(A))))
Fin 234bicolore
```

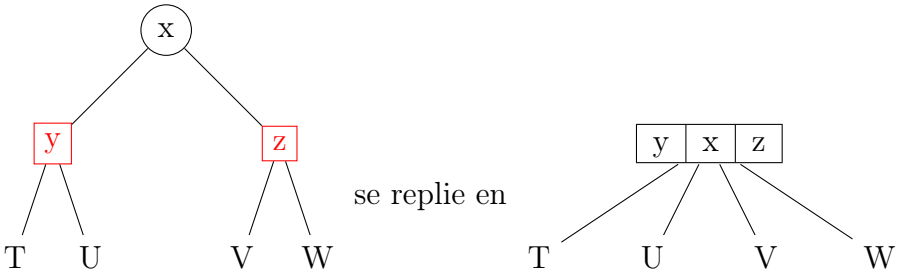
Exercice 1.3 ***: Transformation d'un arbre bicolore en arbre 2-3-4***

Solution

1. L'arête d'un nœud rouge à son père correspond à un pli de remontée : le nœud rouge va prendre place avec son père dans un 3- ou 4-nœud.

Par exemple si on replie ainsi `B-ex1`, on obtient à nouveau `A-ex1`.





2. On aurait aussi pu avoir trois constructeurs pour chacun des types de nœuds des arbres 2-3-4, ce qui aurait été plus précis.

```
bicolore234 (A)
  si EstFeuilleBlanche (A) alors retourne (ArbreVide234())
  LR <- [Clef (A)]
  LF <- []
  G <- SousArbreGauche(A)
  D <- SousArbreDroit(A)
  si non (EstFeuilleBlanche (G))
     alors si Couleur (G)=rouge
           alors LR <- Clef (G) + LR
                 LF <- [bicolore234(SousArbreGauche(G)),
                              bicolore234(SousArbreDroit(G))] + LF
           sinon LF <- [bicolore234(G)] + LF
  si non (EstFeuilleBlanche (D))
     alors si Couleur (D)=rouge
           alors LR <- LR + Clef (SousArbreDroit(A))
                 LF <- LF + [bicolore234(SousArbreGauche(D)),
                                    bicolore234(SousArbreDroit(D))]
           sinon LF <- LF + [bicolore234(G)]
  retourne (Arbre234 (LR, LF))
```

Exercice 1.4 *: Rotations dans un arbre binaire*

Solution

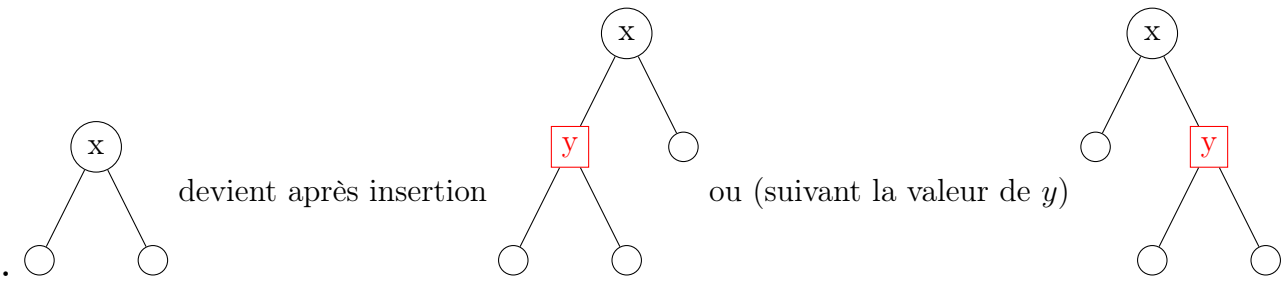
1. Voir les dessins du cours. Ci-dessous un pseudocode pour un des cas

```
RGD(A)
  si EstVide(A) ou EstVide(SousArbreGauche(A))
      ou EstVide{SousArbreDroit(SousArbreDroit(A))}
  alors retourne (A)
  sinon
    p <- Racine(SousArbreGauche(A))
    q <- Racine(SousArbreDroit(SousArbreGauche(A)))
    r <- Racine(A)
    T <- SousArbreGauche(SousArbreGauche(A))
    U <- SousArbreGauche(SousArbreDroit(SousArbreGauche(A)))
    V <- SousArbreDroit(SousArbreDroit(SousArbreGauche(A)))
    W <- SousArbreDroit(A)
    retourne (ArbreBinaire(q,ArbreBinaire(p,T,U),ArbreBinaire(r,V,W)))
  fin si
fin RGD
```

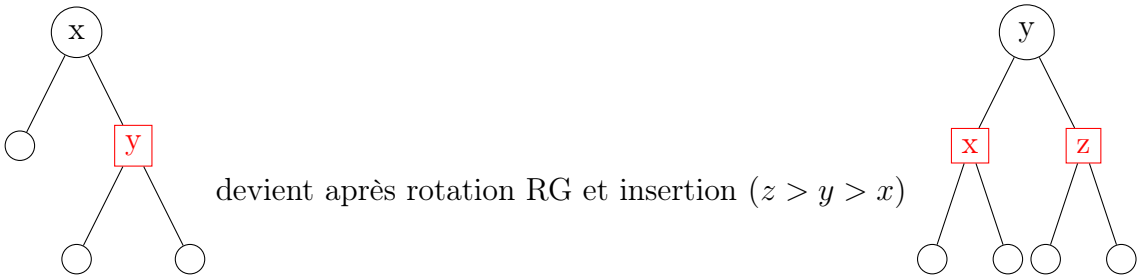
Exercice 1.5 *: Insertion dans un arbre bicolore*

Solution

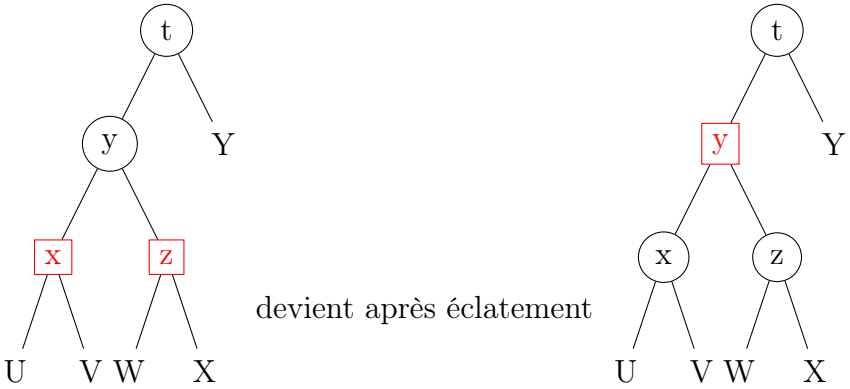
1.



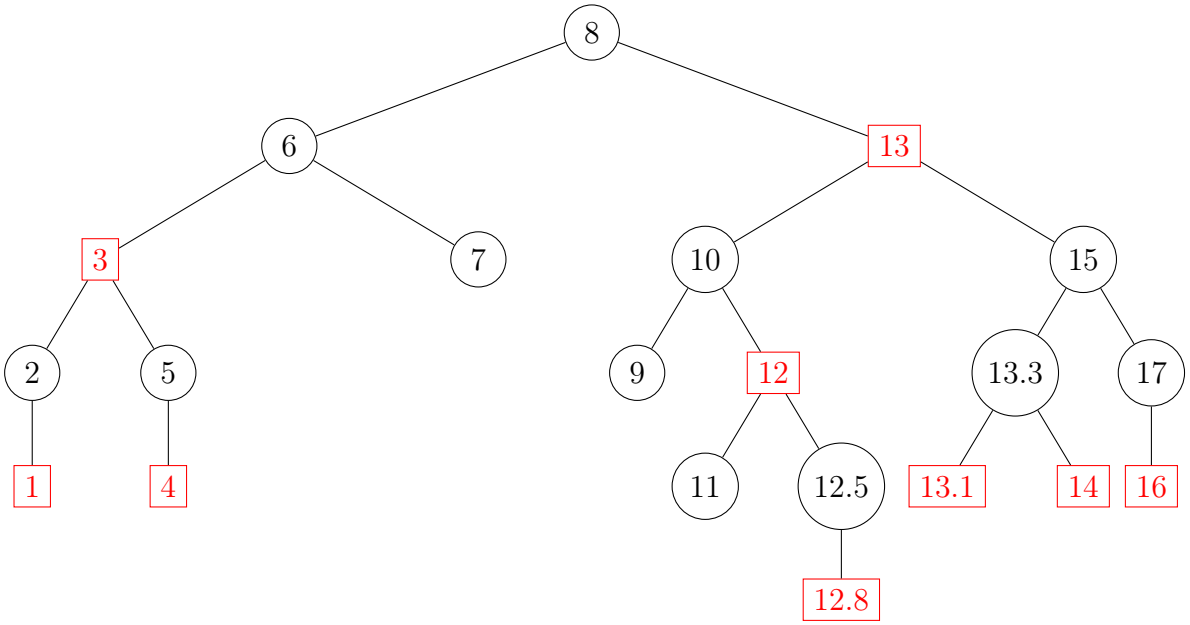
Tous les cas du 3-nœud ne sont pas traités :



2. Tous les cas d'éclatement ne sont pas traités :



3.



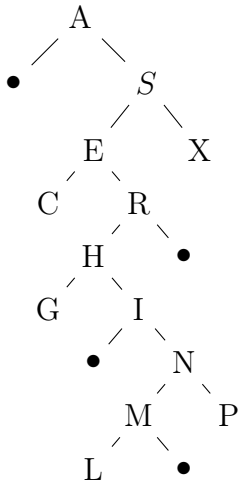
4. On peut ajouter une fonction auxilaire qui teste si la racine est un 4-noeud.
Il faut un programme principal pour éventuellement remettre la racine à blanc (cas d'éclatement) qui lance la fonction récursive où l'éclatement se fera par inversion des couleurs. Beaucoup de cas à étudier pour écrire complètement la fonction. Inutile de tous les traiter.

5. Éclatements à la remontée au lieu de la descente...

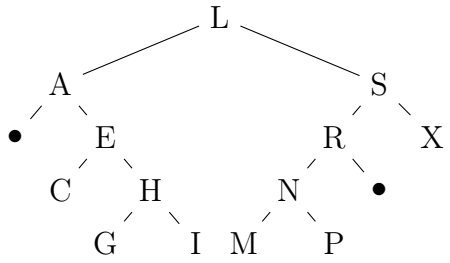
2 Arbres de Recherche versus Tries

Exercice 2.1 ***: Comparaisons sur des exemples***
Solution

1.

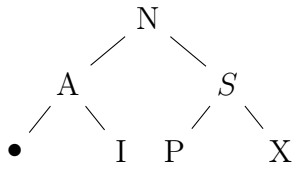


2.

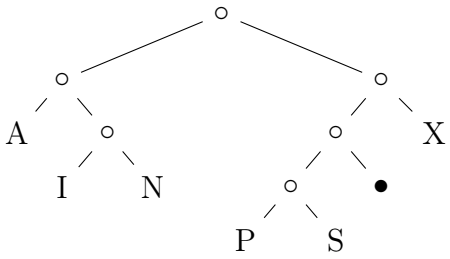


3. L'ordre d'insertion est important dans l'ABR et l'arbre digital. Il n'importe pas dans l'arbre lexicographique.

4. Pour l'ABR et l'arbre digital on trouve le même arbre.



Pour l'arbre lexicographique, ou trie binaire, on obtient :

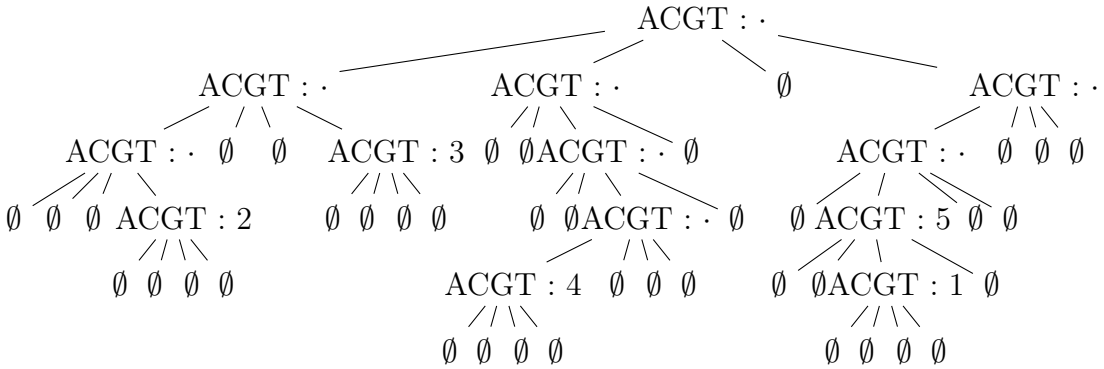


Exercice 2.2 ***: R-trie ou arbres de la Briandais ?***

Solution

1. Il faut dessiner un peu...

2.



3. Il faut dessiner un peu...

4. On a besoin de
- — BRD : caractère, BRDarbre, BRDarbre → BRDarbre : renvoie le BRDarbre correspondant
- — pour un BRDarbre A, on a les les fonctions clef(A), fils(A), frere(A) qui renvoient resp. la clef, et des BDR.
- — Arbre vide : $\emptyset$ ou Nil

5. Rappel : les lettres "frères" sont triées dans l'ordre alphabétique (avec ϵ le premier caractère).
Les fonctions `tete` et `queue` renvoient respectivement le premier caractères et tout le mot privé du premier caractère d'une chaîne de caractères.
On commence par définir une fonction construisant un BRD à partir d'un mot.

```
BRDcons(m)

  si m = '' alors
    renvoyer BRD(epsilon, Nil, Nil)
  sinon
    renvoyer BRD(tete(m), BRDcons(queue(m)), Nil)

fin BRDcons
```

Pour la fonction d'insertion, on suppose que le mot à insérer ne figure pas encore dans l'arbre.

```
BRDinsertion(A,m)

  si m = '' alors
    renvoyer BRD(epsilon, Nil, A)
  si A = Nil alors
    renvoyer BRDcons(m)

  t = tete(m)
  si clef(A) < t alors
    Fr = BRDinsertion(frere(A), m)
    renvoyer BRD(clef(A), fils(A), Fr)

  si clef(A) = t alors
    Fi = BRDinsertion(fils(A), queue(m))
    renvoyer BRD(clef(A), Fi, frere(A))

  si clef(A) > t alors
    renvoyer BRD(t, BRDcons(queue(m)), A)

fin BRDinsertion
```

6. On peut faire appel à une nouvelle primitive : BRDnbMots : BRDarbre → entier : renvoie le nombre de mots encodés dans l'arbre.

```
BRDsuppression(A, m)

  si m = '' alors
    renvoyer frere(A)

  si BRDnbMots(A) = 1 alors
    renvoyer Nil

  t = tete(m)
  si clef(A) < t alors
    renvoyer BRD(clef(A), fils(A), BRDsuppression(frere(A), m))
  sinon
    renvoyer BRD(clef(A), BRDsuppression(fils(A), queue(m)), frere(A))

fin BRDsuppression
```

Exercice 2.3 ***: PATRICIA Trie***

Solution

1.

2. On a besoin de
— PATvide : $\rightarrow$ PATarbre : renvoie un noeud vide avec tous les fils à Nil

3. PATcons prend un mot et construit l'arbre correspondant à ce mot. Attention, PATcons, ne rajoute pas le caractère epsilon !

```
PATcons(m)

  A = PATvide()
  A.cle(tete(m)) = m

fin PATcons
```

On suppose que A ne contient pas m

```
PATinsertion(A, m)

  t = tete(m)
  c = A.cle(t)
  si c = empty alors
    A.cle(t) = concat(m, 'epsilon')
    renvoyer A

  si estPrefixe(c, m) alors
      A.fils(t) = PATinsertion(A.fils(t), m[lg(c)+1 .. lg(m)])
    renvoyer A

  p = prefixe(c, m)         #le plus long prefixe commun
  F = PATcons(c[lg(p)+1 .. lg(c)])
  F.fils(c[lg(p)+1]) = A.fils(tete(c))
  n = m[lg(p)+1 .. lg(m)]
  F.cle(tete(n)) = concat(n, 'epsilon')
  A.cle(t) = p
  A.fils(t) = F
  renvoyer A

fin PATinsertion
```

4. On suppose que m est dans A Avancer dans l'arbre en avançant dans m (un peu à la manière de l'arbre de La Briandais).
Attention quand récursivement une clef passe à Nil, il peut ne rester dans le noeud où elle se situait qu'une seule clef. Dans ce cas, il faut remonter cette clef au niveau de son père.

5. Réfléchir à ce qui se passe s'il y a des mots en commun.