Arbres 2-3-4 (2-3-4 trees)
Correction

1 Préliminaires

Solution 1.2 **(Arbres 234 : Propriétés et représentation)**

2. *Type de données représentant les arbres 2-3-4 :*

```
constantes
    Nbelts    = 3
types
     /* déclaration du type t_ element */
    t_a234    = ↑ noeud_234
    tab3cles  = Nbelts t_element
    tab4fils  = (Nbelts+1) t_a234
    noeud_234 = enregistrement
        entier    nbcles
        tab3cles  cle
        tab4fils  fils
    fin enregistrement noeud_234
```

Remarque : les pointeurs vers les k premiers fils sont à NUL pour les "k-feuilles".

Solution 1.3 **(Minimum et maximum)**

2. **Spécifications :**
   La fonction `max_a234 (A)` retourne la clé maximum de l'arbre 2-3-4 non vide A.

```
algorithme fonction max_a234 : t_element
    parametres locaux
        t_a234    A

debut
    tant que A↑.fils[1] <> NUL faire
        A ← A↑.fils[A↑.nbcles+1]
    fin tant que
    retourne A↑.cle[A↑.nbcles]
fin algorithme fonction max_a234
```

Spécifications :
La fonction `min_a234 (A)` retourne la clé minimum de l'arbre 2-3-4 non vide A.

```
algorithme fonction min_a234 : t_element
    parametres locaux
        t_a234    A

debut
    tant que A↑.fils[1] <> NUL faire
        A ← A↑.fils[1]
    fin tant que
    retourne A↑.cle[1]
fin algorithme fonction min_a234
```

Solution 1.4 **(Recherche d'un élément – *contrôle nov. 12*)**

Spécifications :

La fonction `search234 (t_element` x`, t_a234` A`)` retourne un pointeur vers le nœud contenant la valeur x dans l'arbre A ou la valeur NUL si x n'est pas présent dans l'arbre.

1. *Version récursive :*

```
algorithme fonction search234 : t_a234
    parametres locaux
        t_element    x
        t_a234       A

    variables
        entier    i
debut
    si A = NUL alors
        retourne NUL
    sinon
        i ← 1
        tant que (i <= A↑.nbcles) et (x > A↑.cles[i]) faire
            i ← i+1
        fin tant que
        si (i <= A↑.nbcles) et (x = A↑.cles[i]) alors
            retourne A
        sinon
            retourne recherche (x, A↑.fils[i])
        fin si
    fin si fin algorithme fonction search234
```

2. *Version itérative :*

```
algorithme fonction search234_iter : t_a234
    parametres locaux
        t_element    x
        t_a234       A

    variables
        entier    i
debut
    tant que A <> NUL faire
        i ← 1
        tant que (i <= A↑.nbcles) et (x > A↑.cles[i]) faire
            i ← i+1
        fin tant que
        si (i <= A↑.nbcles) et (x = A↑.cles[i]) alors
            retourne A
        sinon
            A ← A↑.fils[i]
        fin si
    fin tant que
    retourne NUL
fin algorithme fonction search234_iter
```

***Solution 1.5* (Intervalle – *contrôle nov. 12*)**

Spécifications : La procédure `range (A, bi, bs)` affiche (en ordre croissant) l'ensemble des clés se trouvant dans l'arbre 234 A comprises dans l'intervalle $[bi; bs]$. Les clés seront séparées par des espaces.

Remarques :
Pour ne pas *omettre* certaines clés il faut bien faire attention aux propriétés de la relation d'ordre des arbre 2.3.4, et particulièrement :

- Il ne faut pas oublier que même si la dernière clé d'un noeud est inférieure à la borne minimale de l'intervalle, il **peut y avoir des clés dans l'intervalle** dans le sous-arbre issu du dernier fils.
- De même, si la première clé du noeud est supérieure à la borne maximale, il **peut y avoir des clés dans l'intervalle** dans le sous-arbre issu du premier fils.

L'algorithme pourra avoir la structure suivante :

Soient x la première clé supérieure ou égale à bi dans le nœud racine et y le dernière clé inférieure ou égale à bs dans le nœud racine.

Si l'arbre n'est pas vide :
- ◇ Recherche dans le nœud racine de la première clé supérieure ou égal à bi : x
- ◇ Si on est en feuille :
  - ○ Affichage des clés entre bi et bs (de x à y).
- ◇ Sinon
  - ○ Pour toutes les clés de x à y :
    - afficher les clés du fils gauche comprises entre bi et bs
    - afficher la clé
  - ○ afficher les clés du fils droit de y comprises entre bi et bs.

```
algorithme procedure range
   parametres locaux
      t_a234   A
      entier   bi, bs

   variables
      entier   i
debut
   si A <> NUL alors
      i ← 1
      tant que (i <= A↑.nbcles) et (A↑.cle[i] < bi) faire
         i ← i + 1
      fin tant que
      si A↑.fils[1] = NUL alors
         tant que (i <= A↑.nbcles) et (bs >= A↑.cle[i]) faire
            ecrire (A↑.cle[i], " ")
            i ← i + 1
         fin tant que
      sinon
         range (A↑.fils[i], bi, bs)
         tant que (i <= A↑.nbcles) et (bs >= A↑.cle[i]) faire
            ecrire (A↑.cle[i], " ")
            range (A↑.fils[i + 1], bi, bs)
            i ← i + 1
         fin tant que
      fin si
   fin si
fin algorithme procedure range
```

2 Insertions – Suppressions

Solution 2.1 **(Insertion d'un nouvel élément : la méthode classique)**

1. (c) La technique pour résoudre ce problème est l'éclatement (voir la figure 1).

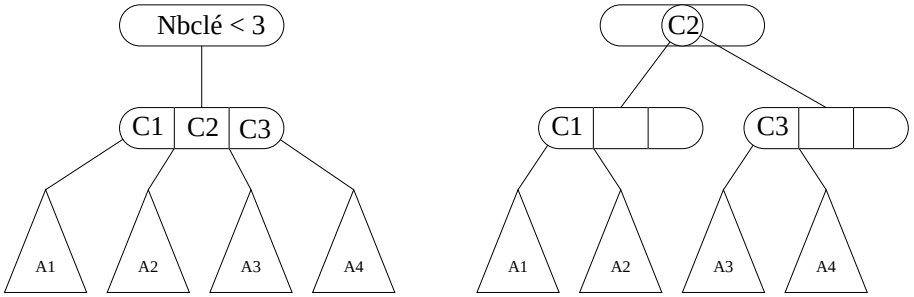

Figure 1 – Éclatement du nœud (C1-C2-C3).

(d) **Spécifications :**
La procédure `eclate (A, i)` éclate le fils n°i de l'arbre A (de type `t_a234`).
– L'arbre A existe et sa racine n'est pas un 4-nœud.
– Le fils i de A existe et sa racine est un 4-nœud.

```
algorithme procedure eclate
    parametres locaux
        t_a234    A
        entier    i

    variables
        t_a234    T          /* pour simplifier ! */
        entier    j
debut
                  /* Création du nouveau nœud */
    allouer(T)
        /* Transfert de la dernière clé du fils i et de ses deux fils vers le nouveau nœud */
    T↑.cle[1] ← A↑.fils[i]↑.cle[3]
    T↑.fils[1] ← A↑.fils[i]↑.fils[3]
    T↑.fils[2] ← A↑.fils[i]↑.fils[4]
    T↑.nbcles ← 1
                  /* Le fils i n'a plus qu'une clé */
    A↑.fils[i]↑.nbcles ← 1

                  /* Insertions dans le nœud père */
    pour j ← A↑.nbcles jusqu'a i decroissant faire
        A↑.cle[j+1] ← A↑.cle[j]
        A↑.fils[j+2] ← A↑.fils[j+1]
    fin pour
    A↑.clé[i] ← A↑.fils[i]↑.cle[2]
    A↑.fils[i+1] ← T
    A↑.nbcles ← A↑.nbcles + 1
fin algorithme procedure eclate
```

2. *Insertion avec éclatement à la descente :*

On recherche le point d'insertion de la nouvelle clé dans l'arbre. Si on n'est pas sur une feuille, avant de descendre sur un fils, on vérifie que sa racine n'est pas un 4-nœud et on l'éclate le cas échéant.

De cette façon, une fois arrivé en feuille, il suffit de placer la clé au bon endroit.

Insertion avec éclatement à la remontée :

On effectue les éclatements des nœuds à la remontée tant qu'on rencontre des 4-nœuds. Les éclatements peuvent ainsi se propager jusqu'à la racine !

3. On insère la clé x dans l'arbre A, sauf si celle-ci est déjà présente.
La version donnée ici sera un algorithme récursif qui prendra donc en paramètre un arbre non vide dont la racine n'est pas un 4-nœud.

Pour traiter le cas des arbres ayant une racine avec trois clés, un premier algorithme (l'algorithme d'appel `insertion_a234`) éclate la racine pour transformer l'arbre dans les hypothèses de l'algorithme récursif. Le cas de l'arbre vide sera aussi traité par cet algorithme.

Spécifications :
La fonction `insert_234 (`x`, A)` insère la clé x dans l'arbre A de type `t_a234`, sauf si celle-ci est déjà présente. L'arbre A n'est pas vide, et sa racine n'est pas un 4-nœud.

```
algorithme fonction insert_234 : booleen
    parametres locaux
        t_element    x
        t_a234       A

    variables
        entier    i, j
debut
                /* Recherche du point d'insertion de x */
    i ← 1
    tant que (i <= A↑.nbcles) et (x > A↑.cle[i]) faire
        i ← i+1
    fin tant que

    si (i <= A↑.nbcles) et (x = A↑.cle[i]) alors     /* x ∈  nœud courant */
        retourne faux
    sinon
        si A↑.fils[1] = NUL alors
                        /* décalage des clés et insertion de la nouvelle clé */
            pour j ← A↑.nbcles jusqu'a i decroissant faire
                A↑.cle[j+1] ← A↑.cle[j]
            fin pour
            A↑.nbcles ← A.nbcles + 1
            A↑.fils[A↑.nbcles+1] ← NUL
            A↑.cle[i] ← x
            retourne vrai
        sinon
            si A↑.fils[i]↑.nbcles = 3 alors
                si A↑.fils[i]↑.cle[2] = x alors
                    retourne faux
                fin si
                eclate (A, i)
                si x > A↑.cle[i] alors
                    i ← i + 1
                fin si
            fin si
            retourne insert_234 (x, A↑.fils[i])
        fin si
    fin si
fin algorithme fonction insert_234
```

Spécifications :
La fonction `insertion_a234 (x, A)` insère la clé x dans l'arbre A de type `t_a234`. Elle retourne un booléen indiquant si l'insertion a eu lieu.

```
algorithme fonction insertion_a234 : booleen
    parametres locaux
        t_element    x
    parametres globaux
        t_a234       A

    variables
        entier     i
        t_a234     T
debut
    si A = NUL alors
        allouer(A)
        A↑.nbcles ← 1
        A↑.cle[1] ← x
        A↑.fils[1] ← NUL
        A↑.fils[2] ← NUL
        retourne vrai
    sinon
        si A↑.nbcles = 3 alors
            allouer (T)
            T↑.nbcles ← 0
            T↑.fils[1] ← A
            A ← T
            eclate (A, 1)
        fin si
        retourne insert_234 (x, A)
    fin si
fin algorithme fonction insertion_a234
```

Solution 2.2 **(Suppression d'un élément : à la descente)**

2. **Rotations**
   (b) Pour simplifier leur utilisation, dans les deux algorithmes de rotations, i représente le numéro du fils qui accueille une nouvelle clé.
   D'autre part, ces algorithmes peuvent être utilisés dans d'autres cas, donc on généralise en ne se limitant pas au cas où le fils qui gagne une clé est un 2-nœud !

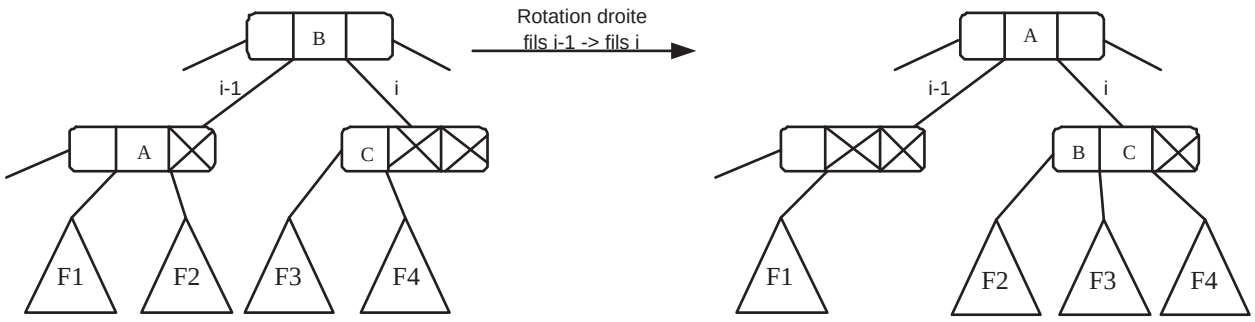

FIGURE 2 – Rotation droite.

Spécifications :
La procédure `rd_gen (A, i)` effectue une rotation du fils $i-1$ vers le fils i (voir figure 2).
Conditions : l'arbre A existe, son fils i existe et sa racine n'est pas un 4-nœud, le fils $i-1$ existe et sa racine n'est pas un 2-nœud.

```
algorithme procedure rd_gen
    parametres locaux
        t_a234    A
        entier    i

    variables
        t_a234    G, D      /* pour simplifier ! */
        entier    j
debut
    G ← A↑.fils[i-1]
    D ← A↑.fils[i]
                              /* décalage des clés et des fils de D */
    pour j ← D↑.nbcles jusqu'a 1 decroissant faire
        D↑.cle[j+1] ← D↑.cle[j]
        D↑.fils[j+2] ← D↑.fils[j+1]
    fin pour
    D↑.fils[2] ← D↑.fils[1]
    D↑.cle[1] ← A↑.cle[i-1]
    D↑.nbcles ← D↑.nbcles + 1

                        /* Déplacement de la dernière clé et du dernier fils de G */
    A↑.cle[i-1] ← G↑.cle[G↑.nbcles]
    D↑.fils[1] ← G↑.fils[G↑.nbcles+1]
    G↑.nbcles ← G↑.nbcles-1
fin algorithme procedure rd_gen
```

Spécifications :
La procédure `rg_gen (A, i)` effectue une rotation du fils $i+1$ vers le fils i.
Conditions : L'arbre A existe, son fils i existe et sa racine n'est pas un 4-nœud, le fils $i+1$ existe et sa racine n'est pas un 2-nœud.

```
algorithme procedure rg_gen
   parametres locaux
      t_a234    A
      entier    i

   variables
      entier    j
      t_a234    G, D      /* pour simplifier ! */
debut
    G ← A↑.fils[i]
    D ← A↑.fils[i+1]
    G↑.cle[G↑.nbcles+1] ← A↑.cle[i]
    G↑.nbcles ← G↑.nbcles + 1
    A↑.cle[i] ← D↑.cle[1]
    G↑.fils[G↑.nbcles+1] ← D↑.fils[1]
                                  /* décalages des clés de D */
    pour j ← 1 jusqu'a D↑.nbcles-1 faire
        D↑.cle[j] ← D↑.cle[j+1]
    fin pour
    si D↑.fils[1] <> NUL alors        /* décalages des fils de D */
        pour j ← 1 jusqu'a D↑.nbcles faire
            D↑.fils[j]← D↑.fils[j+1]
        fin pour
    fin si
    D↑.nbcles ← D↑.nbcles-1
fin algorithme procedure rg_gen
```

3. **Fusion**

   (b) **Spécifications :**

   La procédure `fusion (A, i)` fusionne les fils i et $i+1$ de l'arbre A (voir figure 3).
   Conditions : l'arbre A existe et sa racine n'est pas un 2-nœud, ses fils i et $i+1$ existent et leurs racines sont des 2-nœuds.

```
algorithme procedure fusion
    parametres locaux
        t_a234    A
        entier  i

    variables
        entier  j
        t_a234    G, D      /* pour simplifier ! */
debut
    G ← A↑.fils[i]
    D ← A↑.fils[i+1]
    G↑.cle[2] ← A↑.cle[i]
    G↑.cle[3] ← D↑.cle[1]
    G↑.fils[3] ← D↑.fils[1]
    G↑.fils[4] ← D↑.fils[2]
    G↑.nbcles ← 3
    liberer (D)
    pour j ← i jusqu'a A↑.nbcles-1 faire
        A↑.cle[j] ← A↑.cle[j+1]
        A↑.fils[j+1] ← A↑.fils[j+2]
    fin pour
    A↑.nbcles ← A↑.nbcles -1
fin algorithme procedure fusion
```

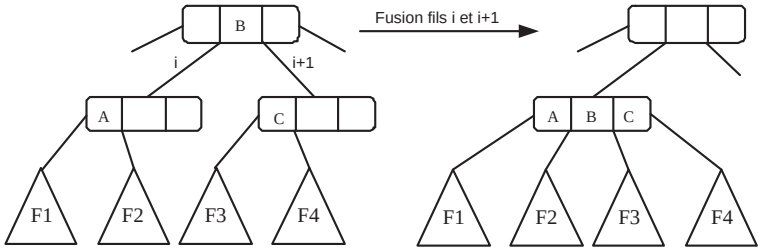

Figure 3 – Fusion

1. (b) *Principe simplifié de suppression dans un arbre 2-3-4 :*

   Nous allons appliquer de nouveau le principe de précaution : les transformations se feront à la descente dès que l'on rencontrera un 2-nœud. On est ainsi sûr de ne jamais avoir deux 2-nœuds qui se suivent, et arrivé sur la feuille, la suppression se fera sans aucun problème.

   - La suppression d'une clé dans un arbre 2-3-4 se fera toujours dans une feuille : si la clé cherchée est dans un nœud interne, il suffit de la remplacer par le maximum de son fils gauche, ou le minimum de son fils droit (qui sont tous deux dans une feuille) et de relancer la suppression de la valeur remontée dans le fils concerné. Le fils choisi doit avoir au moins deux clés en racine (on choisira celui qui en a le plus). Si aucun des deux fils n'a deux clés en racine, on effectue une fusion des deux fils, et on relance la suppression sur le résultat de la fusion.

   - Si le nœud dans lequel on doit descendre est un 2-nœud, il suffit de faire migrer une clé d'un de ses frères (par une rotation) lorsque cela est possible ou de faire une fusion avec l'un de ses frères. La descente peut donc se faire sur un arbre dont la racine n'est pas un 2-nœud.

2. *La suppression :*

Spécifications :

La procédure `suppression_a234_rec (x, A)` supprime la clé x de l'arbre 2-3-4 A. L'arbre A n'est pas vide.

```
algorithme procedure suppression_a234_rec
    parametres locaux
        t_element  x
        t_a234       A

    variables
        entier  i
debut
    i ← 1                                                    /* recherche de la position de x */
    tant que (i <= A↑.nbcles) et (x > A↑.cle[i]) faire
        i ← i+1
    fin tant que

    si A↑.fils[1] <> NUL alors                                              /* si nœud interne */
        si (i <= A↑.nbcles) et (A↑.cle[i] = x) alors      /* on a trouvé x */
            si A↑.fils[i]↑.nbcles > A↑.fils[i+1]↑.nbcles alors
                A↑.cle[i] ← max_a234 (A↑.fils[i])
                suppression_a234_rec (A↑.cle[i], A↑.fils[i])
            sinon
                si A↑.fils[i+1]↑.nbcles > 1 alors
                    A↑.cle[i] ← min_a234 (A↑.fils[i+1])
                    suppression_a234_rec (A↑.cle[i], A↑.fils[i+1])
                sinon
                    fusion(A,i)
                    suppression_a234_rec (x, A↑.fils[i])
                fin si
            fin si
        sinon                                          /* x n'est pas dans le nœud */
            si A↑.fils[i]↑.nbcles = 1 alors
                si (i > 1) et (A↑.fils[i-1]↑.nbcles > 1) alors
                    rd_gen (A,i)
                sinon
                    si (i <= A↑.nbcles) et (A↑.fils[i+1]↑.nbcles > 1) alors
                        rg_gen (A,i)
                    sinon
                        si i>1 alors
                            i ← i-1
                        fin si
                        fusion (A,i)
                    fin si
                fin si
            fin si
            suppression_a234_rec (x, A↑.fils[i])
        fin si
    sinon                                                                   /* si feuille */
        si (i <= A↑.nbcles) et (A↑.cle[i] = x) alors
            pour j ← i jusqu'a A↑.nbcles-1 faire
                A↑.cle[j] ← A↑.cle[j+1]
            fin pour
            A↑.nbcles ← A↑.nbcles-1
        fin si                                  /* sinon x ∉ A */
    fin si
fin algorithme procedure suppression_a234_rec
```

La procédure d'appel : elle se contente de lancer la procédure de suppression et de remplacer la racine par son fils unique en retour si celle-ci est devenue vide (la procédure récursive a effectué une fusion sur la racine qui était un 2-nœud).

Spécifications :
La procédure `suppression_a234 (`x`, `A`)` supprime la clé x de l'arbre 2-3-4 A.

```
algorithme procedure suppression_a234
    parametres locaux
        t_element  x
    parametres globaux
        t_a234       A

    variables
        t_a234    temp
debut
    si A <> NUL alors
        suppression_a234_rec (x, A)
        si A↑.nbcles = 0 alors
            temp ← A
            A ← A↑.fils[1]
            liberer (temp)
        fin si
    fin si
fin algorithme procedure suppression_a234
```

Bonus

Solution 2.3 **(Insertion, une nouvelle méthode : pour changer un peu. . .)**

2. *L'ajout :*

   (c) **Principe de l'insertion :**
   La structure est la même que pour l'insertion "classique" avec principe de précaution vue à l'exercice précédent.

   A la descente vers la feuille, on teste si le nœud par où on doit passer (la racine du fils n° i) contient au plus deux clés. Si ce n'est pas le cas, on va essayer de créer de la place dans ce nœud en examinant ses frères. On effectuera une des trois transformations suivantes avant de continuer à descendre :

   ▷ Une rotation droite :
   – si le frère droit existe, a au plus deux clés et si la dernière clé du fils i (celle qui remonte) est $> x$,
   – ou si le frère droit existe et n'a qu'une seule clé (dans ce cas si l'insertion doit se poursuivre dans le fils droit, cela ne pose pas de problème).

   ▷ Une rotation gauche :
   – si le frère gauche existe, a au plus deux clés et si la première clé du fils i est $< x$,
   – ou si le frère gauche existe et n'a qu'une seule clé.

   ▷ Un éclatement dans les autres cas.

   (d) **Spécifications :**
   La procédure `insertion_a234_rec (`x`, `A`)` insère la clé x dans l'arbre A de type `t_a234`, sauf si celle-ci est déjà présente. L'arbre A n'est pas vide, et sa racine n'est pas un 4-nœud.
   La procédure d'appel qui gère les cas d'insertion dans un arbre vide, et l'éclatement de la racine est la même qu'à l'exercice 2.1.

```
algorithme procedure insertion_a234_rec
    parametres locaux
        t_element   x
        t_a234      A

   variables
     entier    i,j

debut
    i ← 1
    tant que (i <= A↑.nbcles) et (x > A↑.cle[i])) faire
        i ← i+1
    fin tant que

    si (i > A↑.nbcles) ou (x <> A↑.cle[i]) alors  /* x ∉ au nœud courant */

       si A↑.fils[1] = NUL alors                          /* feuille : insertion */
          pour j ← A↑.nbcles jusqu'a i decroissant faire
             A↑.cle[j+1] ← A↑.cle[j]
          fin pour
          A↑.cle[i] ← x
          A↑.nbcles ← A↑.nbcles+1
          A↑.fils[A↑.nbcles+1] ← NUL
       sinon
           si A↑.fils[i]↑.nbcles = 3 alors
               si (i<=A↑.nbcles) et
                   ( ((A↑.fils[i+1]↑.nbcles<3) et (x<A↑.fils[i]↑.cle[A↑.fils[i]↑.nbcles]))
                     ou ( A↑.fils[i+1]↑.nbcles<2) )
               alors
                        rd_gen (i+1)
                        si x > A↑.cle[i] alors
                           i ← i+1
                        fin si
               sinon
                   si (i>1) et ( ((A↑.fils[i-1]↑.nbcles<3) et (x>A↑.fils[i]↑.cle[1]))
                                  ou ( A↑.fils[i-1]↑.nbcles<2) )
                   alors
                        rg_gen (i-1)
                        si x <= A↑.cle[i-1] alors
                            i ← i-1
                        fin si
                   sinon
                        eclate (A,i)
                        si x > A↑.cle[i] alors
                            i ← i+1
                        fin si
                   fin si
               fin si
               si non ((i <= A ↑.nbcles) et (x = A↑.cle[i])) alors
                    insertion_a234_rec (x, A↑.fils[i])
               fin si
       fin si
    fin si
fin algorithme procedure insertion_a234_rec
```

Solution 2.4 **(Suppression d'un élément : à la remontée !)**

1. *Suppressions :*
   La solution consiste à effectuer la suppression en descendant sans se préoccuper des 2-nœuds, et effectuer des modifications en remontant, seulement lorsque cela est nécessaire.

2. **Le principe de suppression :**

   On descend en feuille à la recherche de la clé à supprimer :
   – Si la clé est présente dans un nœud interne, on remplace celle-ci par le maximum de son fils gauche, on relance la suppression sur cette valeur.
   On pourrait ici appliquer le même principe que précédemment, c'est à dire remplacer par le minimum du fils droit si sa racine a plus de clés que celle du fils gauche !
   – Si la clé est dans une feuille, on la supprime.

   En remontant on regarde si on a engendré un fils vide (avec aucune clé). Si c'est le cas, **deux cas sont possibles :**

   cas 1 On fait une rotation en utilisant un frère qui a au moins 2 clés :
   (a) le frère droit existe et contient au moins deux clés
   ⇒ on utilise une **rotation gauche** pour ajouter une clé dans le nœud vide,
   (b) sinon, le frère gauche existe et contient au moins deux clés
   ⇒ on utilise une **rotation droite** pour ajouter une clé dans le nœud vide.

   cas 2 On fait une **fusion** du nœud vide avec un de ses frères pour former un nœud avec deux clés.

   Cette dernière transformation (la fusion) enlève une clé dans le nœud courant, si celui-ci devient vide, il faut donc traiter le cas en remontant de la même manière. On utilise pour cela une fonction qui retourne un booléen indiquant qu'elle a engendré un nœud vide.

   A noter que les algorithmes de rotations et de fusion doivent être modifiés : on peut les généraliser en ajoutant pour le fils i concerné (celui que l'on veut compléter, ainsi que son frère gauche dans le cas d'une fusion) la possibilité d'avoir 1 ou 0 clé.

3. La gestion de la racine de l'arbre se fera tout simplement en testant si le retour de la suppression à généré un nœud vide en racine. Il suffira alors de faire une fusion.

   Spécifications :
   La procédure `del_a234 (`x`, `A`)` supprime la clé x de l'arbre 2-3-4 A.

```
algorithme procedure del_a234
    parametres locaux
        entier    x
    parametres globaux
        t_a234    A

    variables
        t_a234    T
debut
    si A <> NUL alors
        si del_rec_a234 (x, A) alors
            T ← A
            A ← A↑.fils[1]
            liberer (T)
        fin si
    fin si
fin algorithme procedure del_a234
```

Spécifications :

La fonction `del_a234_rec (x, A)` supprime la clé x de l'arbre 2-3-4 non vide A. Elle retourne un booléen indiquant si la racine de l'arbre est vide (ne contient plus de clés).

```
algorithme fonction del_rec_a234 : booleen
    parametres locaux
        t_element    x
        t_a234       A

    variables
        entier  i, j
debut
   i ← 1
   tant que (i <= A↑.nbcles) et (x > A↑.cle[i]) faire
        i ← i+1
   fin tant que

   si A↑.fils[1] = NUL alors                                    /* feuille */
        si (i <= A↑.nbcles) et (x = A↑.cle[i]) alors     /* on a trouvé x */
            pour j ← i jusqu'a A↑.nbcles-1 faire
                A↑.cle[j] ← A↑.cle[j+1]
            fin pour
            A↑.nbcles ← A↑.nbcles - 1
            retourne (A↑.nbcles = 0)
        sinon                                  /* x n'est pas présent */
           retourne faux
        fin si

   sinon                                                        /* noeud interne */
        si (i <= A↑.nbcles) et (x = A↑.cle[i]) alors     /* on a trouvé x */
            A↑.cle[i] ← max_a234(A↑.fils[i])
            x ← A↑.cle[i]
        fin si

       /* on lance la suppression de x dans le fils i */

        si del_rec_a234 (x, A↑.fils[i]) alors          /* la racine du fils i est vide */
            si (i <= A↑.nbcles) et (A↑.fils[i+1]↑.nbcles > 1) alors
                 RG_gen (A,i)
                 retourne faux
            sinon
                 si (i > 1) et (A↑.fils[i-1]↑.nbcles > 1) alors
                     RD_gen (A,i)
                     retourne faux
                 sinon
                     si i > A↑.nbcles alors
                          i ← i-1
                     fin si
                     fusion_gen (A,i)
                     retourne (A↑.nbcles = 0)
                 fin si
            fin si
        sinon
            retourne faux
        fin si
   fin si
fin algorithme fonction del_rec_a234
```