Algorithmique
Correction Contrôle n° 1

INFO-SPE – EPITA

8 nov. 2010

Solution 2 **(Arbre 2-3-4 : insertions –** ***4 points*****)**

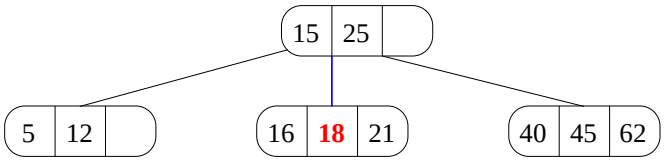

FIG. 1 – Ajout de 18

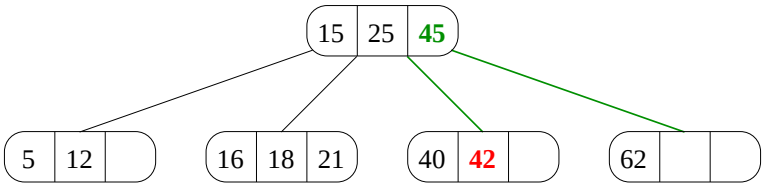

FIG. 2 – Ajout de 42 - Le nœud (40-45-62) a été éclaté

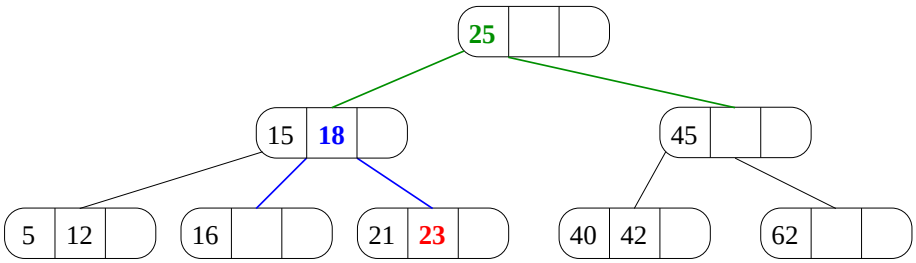

FIG. 3 – Ajout de 23 - La racine, puis le nœud (16-18-21) ont été éclatés

---

Solution 1 **(hachages –** ***7 points*****)**

1. Le principe est simple, on calcule la valeur de hachage primaire de l'élément x à ajouter. On regarde si cette valeur correspond à une case vide, si c'est le cas, on y met le x, on modifie les valeurs d'état et de lien et l'on quitte avec $Vrai$.

   Sinon, Si la case est libre, on mémorise sa position dans lib qui était initialisé à -1.

   Ensuite,on suit le lien jusqu'à le perdre (valeur m) ou bien trouver une case occupé par x. Dans le deuxième cas, on quitte en renvoyant $Vrai$.

   En chemin, on mémorise la première case libre rencontrée.

   Ensuite, Soit on a trouvé sur le chemin une case libre, alors on l'utilise en mettant le x dedans et l'on quitte en renvoyant $Vrai$, soit on cherche à partir de la réserve une case vide. Si l'on en trouve une ($r \geq 0$), on met le x dedans et on quitte en renvoyant $Vrai$.

   Sinon, c'est que le tableau de hachage est plein et l'on quitte en renvoyant $Faux$.

2. Sur cet algorithme, le tableau de hachage est indicé de 0 à $m-1$, il est simple de le transformer pour qu'il fonctionne de 1 à m.

```
Algorithme Fonction ajouter_HCO : Booléen
Paramètres globaux
    t_hachage    th
Paramètres locaux
    t_element x
Variables
    entier  r, i, lib
Debut
    i ← h(x)            /* calcul de la valeur de hachage primaire */
    lib ← -1            /* Position de la première case libre */
    Si etatEst(Th,i,vide) Alors      /* Ajout de l'élément avec modification du lien*/
        th[i].elt ← x
        th[i].etat ← occupee
        th[i].lien ← m
        Retourne(Vrai)
    Fin si
    /* Recherche de x ou d'un lien terminal */
    Tant que i<>m faire
        Si (etatEst(th,i,occupee) et th[i].elt=x) Alors
           Si lib>=0 Alors             /* si case libre, on rapproche x */
              th[lib].elt ← x
              th[lib].etat ← occupee
              th[i].etat ← libre
           Fin si
           Retourne (Vrai)
        Fin si
        Si etatEst(th,i,libre) et lib=-1 Alors  /* Mémorisation de la case libre*/
           lib ← i
        Fin si
        r ← i                  /* Mémorisation de la case précédente */
        i ← th[i].lien
    Fin tant que
    /* récupération de la dernière case i valide */
    i ← r
    Si lib>=0 Alors             /* si case libre, on rapproche x */
       th[lib].elt ← x
       th[lib].etat ← occupee
       Retourne(Vrai)
    Sinon
       /* Recherche de la 1ère place vide ou libre*/
       r ← m-1           /* Position de réserve virtuelle */
       Tant que r>=0 et non(etatEst(th,r,vide)) Faire
           r ← r-1
       Fin tant que
       Si r>=0 Alors                  /* Ajout de l'élément sans modification du lien*/
          th[r].elt ← x
          th[r].etat ← occupee
          th[r].lien ← m
          th[i].lien ← r
          Retourne(Vrai)
       Fin si
       Retourne(Faux)                 /* Tableau plein */
    Fin si
Fin Algorithme Fonction ajouter_HCO
```

Solution 3 **(Arbre 2-3-4 : Des croissants –** ***4 points*****)**

Spécifications : la fonction `decroissant_234 (A)` retourne la chaîne contenant la liste des clés en ordre décroissant de l'arbre A de type `t_a234`.

```
algorithme fonction decroissant_234 : chaine
    parametres locaux
        t_a234    A
    variables
        entier  i
        chaine  s
debut
    si A = NUL alors
        retourne ""
    sinon
        s ← ""
        pour i ← A↑.nbcles jusqu'a 1 decroissant
            s ← s + decroissant_234 (A↑.fils[i+1]) + A↑.cle[i] + ','
        fin pour
        s ← s + decroissant_234 (A↑.fils[1])
        retourne s
    fin si
fin algorithme fonction decroissant_234
```

Remarque : il serait intéressant de faire un cas à part des feuilles pour ne pas rappeler sur les fils vides dans ce cas. Cela permettra de plus de ne pas avoir une ',' de trop à la fin de la chaîne !

```
algorithme fonction decr : chaine
    parametres locaux
        t_a234    A
    variables
        entier  i
        chaine  s
debut
    s ← ""
    si A↑.fils[1] = NUL alors
        pour i ← A↑.nbcles jusqu'a 2 decroissant faire
            s ← s + A↑.cle[i] + ','
        fin pour
        s ← s + A↑.cle[1]
    sinon
        pour i ← A↑.nbcles jusqu'a 1 decroissant faire
            s ← s + decr (A↑.fils[i+1]) + ',' + A↑.cle[i] + ','
        fin pour
        s ← s + decr (A↑.fils[1])
    fin si
    retourne s
fin algorithme fonction decr

algorithme fonction decroissant_234 : chaine
    parametres locaux
        t_a234    A
    variables
        entier  i
        chaine  s
debut
    si A = NUL alors
        retourne ""
```

```
        sinon
            retourne decr (A)
        fin si
    fin algorithme fonction decroissant_234
```

Solution 4 **(Arbres généraux : Préfixe - Suffixe –** ***7 points*****)**

1. **Principe :** On effectue un parcours profondeur classique, en traitement préfixe on incrémente le compteur `c` et on met la clef de `A` dans `v[c]` et en traitement suffixe on incrémente `c` et on remet la clef de `A` dans la case `v[c]`.

```
algorithme procedure ps_nuplet
  parametres locaux
    t_arbre_nuplets          A
  parametres globaux
    entier                   c
    t_vect_cles              v
  variables
    entier                   i
debut
  c ← (c + 1)
  v[c] ← A↑.cle
  pour i ← 1 jusqu'a A↑.nbfils faire
    ps_nuplet(A↑.fils[i], c, v)
  fin pour
  c ← (c + 1)
  v[c] ← A↑.cle
fin algorithme procedure ps_nuplet
```

2.
```
algorithme fonction remplissage_nuplet : entier
  parametres locaux
    t_arbre_nuplets          A
  parametres globaux
    t_vect_cles              v
  variables
    entier                   c
debut
  c ← 0
  ps_nuplet(A, c, v)
  retourne ((c / 2))
fin algorithme fonction remplissage_nuplet
```

3.
```
algorithme procedure ps_dyn
  parametres locaux
    t_arbre_dyn              A
  parametres globaux
    entier                   c
    t_vect_cles              v
  variables
debut
  si (A <> NUL) alors
    c ← (c + 1)
    v[c] ← A↑.cle
    ps_dyn(A↑.fils, c, v)
    c ← (c + 1)
    v[c] ← A↑.cle
    ps_dyn(A↑.frere, c, v)
  fin si
fin algorithme procedure ps_dyn
```