Algorithmique
Contrôle n° 1

Info-Spé – Epita

D.S. 314019.4 BW (7 déc. 2009 - 11 :15)

Consignes (à lire) :

- Vous devez répondre sur **les feuilles de réponses prévues à cet effet**.
  - Aucune autre feuille ne sera ramassée (gardez vos brouillons pour vous).
  - Répondez dans les espaces prévus, **les réponses en dehors ne seront pas corrigées** : utilisez des brouillons !
  - Ne séparez pas les feuilles à moins de pouvoir les ré-agrafer pour les rendre.
  - Aucune réponse au crayon de papier ne sera corrigée.
- La présentation est notée en moins, c'est à dire que vous êtes noté sur 20 et que les points de présentation (2 au maximum) sont retirés de cette note.
- **Les algorithmes :**
  - Tout algorithme doit être écrit dans le langage Algo (pas de C, Caml ou autre).
  - Tout code Algo non indenté ne sera pas corrigé.
  - Tout ce dont vous avez besoin (types, routines) est indiqué en **annexe** (dernière page) !
- Durée : 1h30

Exercice 1 (Hachages – *9 points*)

On considère un ensemble de clés que l'on veut stocker dans une table de hachage de taille m = 11.

Supposons, pour chaque clé, la fonction de hachage suivante qui :
- attribue à chaque lettre de la clé sa valeur ordinale : a→1, b→2, . . . , z→26
- fait la somme des valeurs de toutes les lettres de la clé
- effectue le modulo m de la valeur obtenue au point précédent

Exemple : h(satriani)=(19+1+20+18+9+1+14+9) mod 11 = 3

1. Donner sous forme d'un tableau les valeurs de hachage associées à l'ensemble E suivant :
   E={beck, cale, clapton, hendrix, hooker, king, richards, vaughan, winter, young}

2. Représenter (dessiner) les structures de données et la gestion des collisions pour l'ajout de toutes les clés de l'ensemble E dans les cas suivants :

   (a) Hachage linéaire avec un coefficient de décalage $d = 3$.
   Rappelons la méthode. Soit une fonction de hachage h, la suite de m essais est réalisée comme suit :

$$\begin{aligned} essai_1(x) &= h(x) \\ essai_2(x) &= h(x) \oplus d \\ &\dots \\ essai_i(x) &= h(x) \oplus d(i-1) \\ &\dots \\ essai_m(x) &= h(x) \oplus d(m-1) \end{aligned}$$

   (b) Hachage avec chaînage séparé.

3. Soient les déclarations suivantes :

```
Constantes
  m = 11
Types
  /* déclaration du type t_element */
  t_element = ...
  t_hachage = m t_element
Variables
  t_hachage th

/* Renvoie vrai si la i^ième  place d'un tableau Th est vide, faux sinon */
Algorithme fonction estvide : booléen
Paramètres locaux
  t_hachage   th
  entier      i    /* Avec  0<=i<=m-1 */

/* Calcule la valeur de hachage primaire d'un élément x */
Algorithme fonction h : Entier
Paramètres locaux
  t_element  x
```

Pour le principe et l'algorithme qui suivent, nous n'envisagerons pas la possibilité d'implémenter la suppression. Nous considérerons donc une case du tableau comme ne pouvant avoir que deux états : "Vide" ou "Non vide".

Soit un entier m et th un tableau d'indices de $0..m-1$. Soit h une fonction de hachage définie sur le type des éléments de th et à résultat dans $[0, m-1]$. La méthode dite de ***hachage ordonné*** utilise à la fois le hachage linéaire et des comparaisons entre les valeurs des éléments. C'est à dire que lorsque des éléments sont en collision, ils sont triés par ordre croissant.

   (a) Ecrire le principe d'une fonction booléenne de recherche d'un élément x dans le tableau th selon le principe du *hachage ordonné*.

   (b) En utilisant ce principes et les déclarations précédentes, écrire l'algorithme de la fonction booléenne ***rechercher_HO*** correspondant à la recherche d'un élément x dans le tableau de hachage th selon le principe du *hachage ordonné*.

Exercice 2 (Mesure sur les arbres *2-3-4* – *11 points*)

Dans cet exercice on se propose de mesurer la *qualité* d'un arbre *2-3-4* pour la recherche. L'une des meilleures mesures consiste à évaluer le nombre moyen de clefs par nœud, plus celui-ci est proche de 3 meilleur est l'arbre. À partir de cette mesure on va comparer les deux techniques d'insertion vu en TD.

Le principe de l'insertion repose sur la notion de *principe de précaution* : on s'assure que le nœud sur lequel on descend n'est pas plein. Ainsi, s'il s'agit d'une feuille l'insertion se fera sans problème et sinon, il y aura la place pour un éventuel éclatement d'un de ses fils. Les deux méthodes d'insertion diffèrent sur la façon d'assurer cette pré-condition : la première utilise systématiquement l'éclatement, tandis que la seconde utilise les rotations de préférence aux éclatements.

On rappelle les deux méthodes d'insertion :
- **Éclatements :** avant de descendre, si le nœud cible est plein on l'éclate.
- **Avec rotations :** avant de descendre, si le nœud cible est plein, on essaye d'abord d'utiliser une rotation plutôt qu'un éclatement. On pose : `i` est l'indice du fils cible dans le nœud père (noté `A`) et `x` est la clef à insérer. On distingue les cas suivants :
  1. **Rotation gauche** : le voisin de gauche (d'indice `i-1`) existe (`i>1`), il a au plus deux clefs et, s'il a deux clefs exactement, la clef qui remontera dans `A` (la première clef du fils `i`) n'est pas plus grande que `x`.
  2. **Rotation droite** : la rotation gauche n'est pas possible, le voisin de droite (d'indice `i+1`) existe, il a au plus deux clefs et, s'il a deux clefs exactement, la clef qui remontera dans `A` (la dernière clef du fils `i`) n'est pas plus petite que `x`.
  3. **Éclatement :** si aucune des rotations n'est possible.

1. Expliquez en quelques mots très simples (ce n'est pas un principe) comment on peut calculer le nombre moyen de clefs par nœud.

2. Écrire la fonction `occupation(A)` qui calcule le nombre moyen de clefs par nœud. La fonction `occupation(A)` utilisera la procédure récursive `occup_rec` dont vous donnerez les spécifications. Attention, votre fonction récursive devra être *efficace* (éviter les doubles traversées sur l'arbre ...)

3. Dessiner l'arbre final résultant des insertions dans l'arbre de la figure 1, avec la méthode *éclatements à la descente*, les clefs suivantes : `42, 18, 19`.

4. Dessiner l'arbre final résultant des insertions dans l'arbre de la figure 1, avec la méthode *utilisant les rotations*, les clefs suivantes : `42, 18, 19`.

5. Calculer le taux d'occupation avec l'algorithme de la question 2 sur l'arbre de la figure 1 avant insertions et sur les arbres obtenus aux questions 3 et 4.

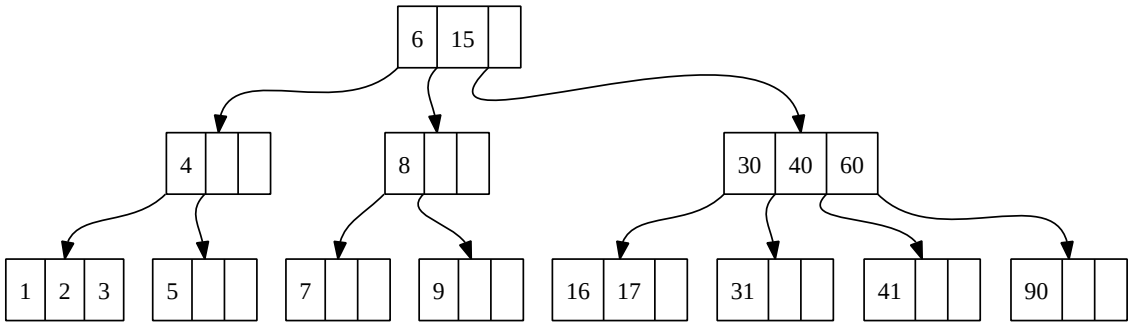

FIG. 1 – Un arbre *2-3-4*

Annexe

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

```
constantes
    degre = 2
types
     /* déclaration du type t_ element */
    t_a234    = ↑ t_noeud_234
    tab3cle   = (2*degre-1) t_element
    tab4fils  = (2*degre) t_a234
    t_noeud_234 = enregistrement
        entier    nbcles
        tab3cle   cle
        tab4fils  fils
    fin enregistrement t_noeud_234
```