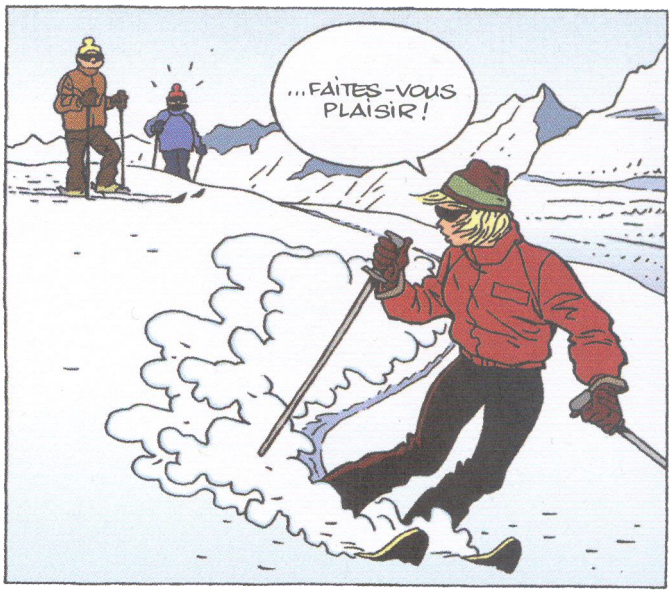
Algorithmique
Contrôle n° 1

S3
EPITA

D.S. 309858.67 BW (10 nov 2014 - 10 :00)

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 : 2h00



Exercice 1 (Quelques résultats différents – *6 points*)

1. Considérant que les collisions sont résolues par hachage coalescent, représenter la table de hachage de 11 éléments correspondant à l'application de la fonction de hachage

$$h(x) = (2x + 5) \ mod \ 11,$$

sur les clés $12, 44, 13, 88, 23, 94, 11, 39, 20, 16, 5$.

2. En reprenant les éléments de la question précédente et en considérant que les collisions sont résolues par hachage linéaire, représenter la table de hachage de 11 éléments qui en résulte.

3. En reprenant les éléments de la première question et en considérant que les collisions sont résolues par double hachage à l'aide de la deuxième fonction de hachage

$$d(x) = 7 - (x \ mod \ 7)$$

qui combinée avec $h(x)$ donne la table 1 d'essais successifs associés à chaque clé, représenter la table de hachage de 11 éléments qui en résulte.

TABLE 1 – valeurs d'essais successifs

Elt	h(x)	2	3	4	5	6	7	8	9	10	11
12	7	9	0	2	4	6	8	10	1	3	5
44	5	10	4	9	3	8	2	7	1	6	0
13	9	10	0	1	2	3	4	5	6	7	8
88	5	8	0	3	6	9	1	4	7	10	2
23	7	1	6	0	5	10	4	9	3	8	2
94	6	10	3	7	0	4	8	1	5	9	2
11	5	8	0	3	6	9	1	4	7	10	2
39	6	9	1	4	7	10	2	5	8	0	3
20	1	2	3	4	5	6	7	8	9	10	0
16	4	9	3	8	2	7	1	6	0	5	10
5	4	6	8	10	1	3	5	7	9	0	2

Exercice 2 (Arbres Généraux : sérialisation – *6,5 points*)

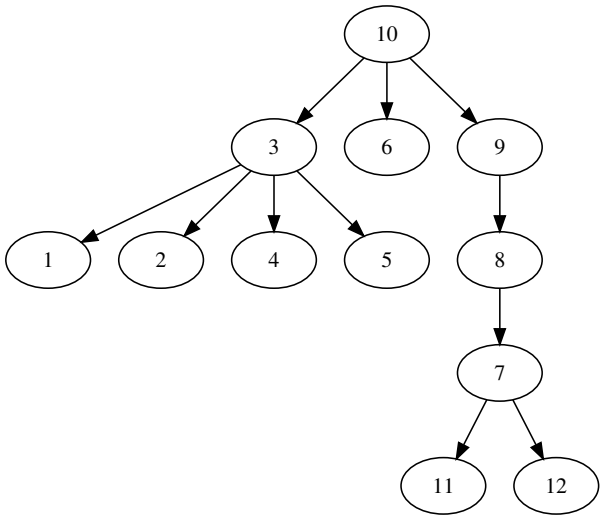


FIGURE 1 – Arbre général

Nous allons nous intéresser à une représentation alternative des arbres généraux : les vecteurs de pères. Cette représentation est linéaire et peut donc être utilisée pour stocker notre arbre dans un fichier (sérialisation).

Le principe est simple : à chaque nœud de l'arbre on associe un identifiant unique, sous la forme d'un entier compris entre 1 et la taille de l'arbre. On construit ensuite un vecteur où la case i contient l'identifiant du père du nœud d'identifiant i. La racine de l'arbre aura pour père -1.

1. Remplir le vecteur de pères pour l'arbre de la figure 1
2. Écrire un algorithme qui, à partir d'un arbre en représentation $n-$uplet de pointeurs, remplit le vecteur de pères correspondant.
3. Écrire le même algorithme pour la représentation premier-fils/frère-droit.

Exercice 3 (B-tree or not B-tree... – *6 points*)

On désire vérifier si un B-arbre est bien "ordonné", c'est-à-dire si la relation d'ordre est bien respectée partout. Pour cela vous devez écrire la fonction récursive `test_Btree` qui prendra en paramètre le B-arbre à tester, ainsi que 2 valeurs de type `t_element` représentant les bornes de l'intervalle sur lequel on travaille.

L'appel de votre fonction sera fait de la manière suivante :

```
algorithme fonction test_Btree_ordered : booleen
  parametres locaux
    t_Btree B
debut
  retourne (B = NUL) ou test_Btree (B, −∞, +∞)
fin algorithme fonction test_Btree_ordered
```

Avec $-\infty$ et $+\infty$ les valeurs extrêmes du type `t_element`.

Exercice 4 (B-arbre : suppression – *1,5 points*)

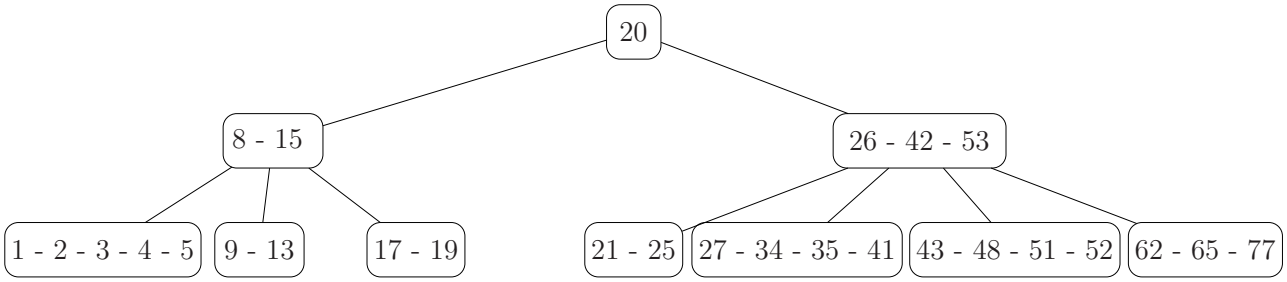


Figure 2 – B-tree

1. L'arbre de la figure 2 est un B-arbre. De quel ordre (degré minimum) est-il ?

2. En utilisant le principe "à la descente", dessiner l'arbre après suppression de la valeur 15.

Annexes

Implémentation des arbres généraux

$n-$uplets de pointeurs :

```
constantes
  Max = ...
types
  t_arbre_nuplet  = ↑t_tuple_node

  t_vect_fils = Max t_arbre_nuplet

  t_tuple_node = enregistrement
    entier        cle
    entier        nbfils
    t_vect_fils    fils
  fin enregistrement t_tuple_node
```

Premier-fils/frère-droit :

```
types
    t_arbre_dyn  = ↑t_dyn_node

    t_dyn_node = enregistrement
      entier    cle
      t_arbre_dyn  fils, frere
    fin enregistre-
ment t_dyn_node
```

Vecteurs d'entiers

```
types
    t_vect_entiers  = Max entier
```

Implémentation des B-arbres

```
constantes
    t    = /* minimal degree */
types
     /* t_element */
    t_Btree    = ↑ t_node_Btree
    t_vect_cles  = (2*t-1) t_element
    t_vect_fils  = (2*t) t_Btree
    t_node_Btree = enregistrement
        entier     nbcles
        t_vect_cles  cles
        t_vect_fils  fils
    fin enregistrement t_node_Btree
```

Rappel : dans le vecteur des fils, les k premiers fils sont à NUL pour les k-nœuds externes.