GIS3 — Structures de Données — Épreuve d'avril 2012
Durée 2 heures — Aucun document autorisé

---

1 Questions de cours (12 points)

1.1 Listes chaînées et langage C

On rappelle le contenu du fichier `liste_double.h`.

```
#if ! defined (LISTE_DOUBLE_H)
#define LISTE_DOUBLE_H 1

/*****************************************************************
 * IMPLANTATION
 *****************************************************************/

struct maillon_double
{   double value;
    struct maillon_double* next;
};

struct liste_double
{   struct maillon_double* tete;
    int nbelem;
};

/*****************************************************************
 * PROTOTYPES DES FONCTIONS (TYPE ABSTRAIT)
 *****************************************************************/

extern void init_liste_double (struct liste_double*);
extern void clear_liste_double (struct liste_double*);
extern void set_liste_double (struct liste_double*, struct liste_double*);
extern void ajouter_en_tete_liste_double (struct liste_double*, double);
extern void extraire_tete_liste_double (double*, struct liste_double*);
extern void imprimer_liste_double (struct liste_double*);
#endif
```

Question 1 [1 pt]. Quelle est l'utilité des directives `#if`, `#define` et `#endif` présentes dans le fichier ?

Question 2 [2 pts]. Donner le code C d'une fonction qui ajoute un `double` en queue d'une liste déclarée comme ci-dessus.

Question 3 [2 pts]. La fonction suivante est incorrecte. Pourquoi ?

```
void clear_liste_double (struct liste_double* L)
{   struct maillon_double* courant;
    int i;

    courant = L->tete;
    for (i = 0; i < L->nbelem; i++)
    {   free (courant);
        courant = courant->next;
    }
}
```

1.2 Files de priorité

On considère le tableau T suivant.

```
int T [] = { 5, 23, 17, 113, 25, 40, 44, 421, 666, 40 };
int n = sizeof (T) / sizeof (int);
```

Question 4 [1 pt]. Est-il un minimier (en supposant qu'un entier a est plus prioritaire qu'un entier b si $a < b$) ? En donner une représentation graphique.

Question 5 [2 pts]. Défiler deux fois de suite un élément. Ensuite, enfiler 4, puis 39. Détailler graphiquement les opérations de restructuration du minimier.

1.3 Tables de hachage

On considère l'insertion des clés 10, 22, 31, 4, 15, 28, 17, 88, 59 dans une table de hachage de $N = 11$ alvéoles. La table est gérée avec la technique du double hachage. La fonction de hachage est $h(s) = (h_1(s), h_2(s)) = (s \bmod N, 1 + s \bmod (N - 1))$.

Question 6 [2 pts]. Insérer les alvéoles dans la table (donner le résultat final uniquement).

Question 7 [1 pt]. Il y avait 9 clés à insérer dans une table de 11 alvéoles. Était-on certain, pour autant, de trouver un alvéole libre pour chaque clé ?

Question 8 [1 pt]. Pour réaliser une table de hachage contenant des chaînes de caractères, un étudiant propose de prendre l'adresse de la chaîne passée en paramètre modulo N. Que pensez-vous de cette idée ?

2 Les geobuckets (8 points)

Fusionner deux listes triées, L_1 et L_2, consiste à construire une nouvelle liste, triée, contenant les éléments des deux listes. Dans le pire des cas, fusionner deux listes triées L_1 et L_2 nécessite $|L_1| + |L_2| - 1$ comparaisons d'éléments (où $|L_1|$ et $|L_2|$ désignent les longueurs des deux listes). Pour simplifier, on suppose que les listes sont sans doublon.

Question 9 [1 pt]. Exprimer, en fonction de n, le nombre de comparaisons d'éléments effectuées par l'algorithme suivant, dans le pire des cas. Justifier en quelques mots.

```
begin
  Soient L_1, L_2, ..., L_n un ensemble de n listes ne contenant chacune qu'un seul élément
  L = ∅
  for i variant de 1 à n do
    L = la liste obtenue en fusionnant L avec L_i
  end do
end
```

Un *geobucket* est une structure de données qui permet de fusionner de façon efficace un grand nombre de listes triées. L'idée ? Éviter de fusionner des listes de longueurs très différentes. Un *geobucket* se présente comme un tableau T de listes triées T_i qui vérifient la propriété suivante :

$$|T_i| = 0 \quad \text{ou} \quad 2^i \leq |T_i| < 2^{i+1}\,. \tag{1}$$

Les indices appartiennent typiquement à l'intervalle [0, 32].

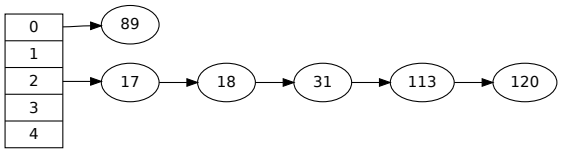
Une opération importante est la **fusion** d'une liste triée L avec un *geobucket* T. On commence par déterminer l'indice i tel que $2^i \leq |L| < 2^{i+1}$, puis on fusionne L et T_i (résultat dans T_i). La nouvelle liste T_i peut être trop longue et ne plus vérifier (1). Dans ce cas, on la fusionne avec T_{i+1} (résultat dans T_{i+1}), puis on vide T_i. La nouvelle liste T_{i+1} peut, elle aussi, être trop longue et ne plus vérifier (1). Dans ce cas, on la fusionne avec T_{i+2} (résultat dans T_{i+2}), puis on vide T_{i+1}. Et ainsi de suite.

L'autre opération importante est la **conversion en liste** d'un *geobucket*. Elle s'effectue en fusionnant toutes les listes T_i, en commençant par les plus petites. Le résultat est une liste triée.

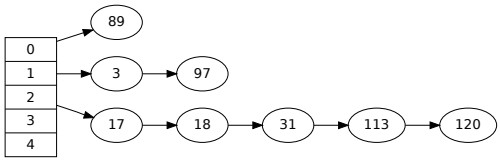
Ces deux opérations sont illustrées par un exemple en Figure 1.

Question 10 [2 pts]. Donner les déclarations C de structures permettant d'implanter le type *geobucket*. Vous pouvez supposer l'existence d'un module de listes.

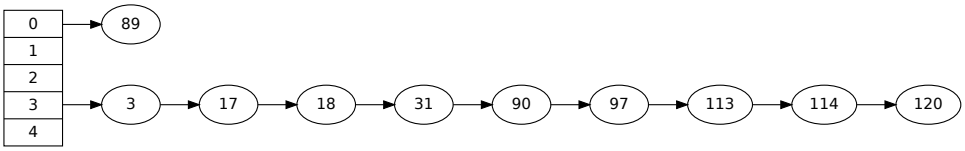
Question 11 [1 pt]. Spécifier le type donné à la question précédente (expliquer les champs et leurs propriétés).



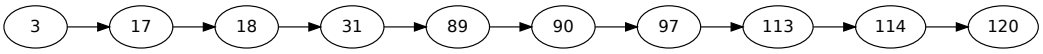
(a)



(b)



(c)



(d)

FIGURE 1 – En (a), un *geobucket* contenant six éléments, répartis dans deux listes : une liste à un élément, qui se trouve (les numéros dans les cases sont les indices de ces cases) en T_0 (en effet, $2^0 \leq 1 < 2^1$), et une liste à cinq éléments, qui se trouve en T_2 (puisque $2^2 \leq 5 < 2^3$). En (b), le résultat de la fusion du *geobucket* (a) avec la liste à deux éléments $L_1 = [3, 97]$. La liste L_1 a été « fusionnée » avec la liste vide en T_1. En (c), le résultat de la fusion du *geobucket* (b) avec la liste à deux éléments $L_2 = [90, 114]$. La liste L_2 a été fusionnée avec T_1. La liste ainsi obtenue, qui faisait 4 éléments, était trop longue pour rester en T_1. Elle a donc été fusionnée, à son tour, avec T_2. La liste ainsi obtenue, qui faisait 9 éléments, était trop longue pour rester en T_2. Elle a été « fusionnée » avec la liste vide en T_3. En (d), la liste triée obtenue après conversion en liste du *geobucket*.

Question 12 [2 pts]. Donner les prototypes des fonctions exportées d'un module minimaliste de *geobuckets*, c'est-à-dire, toutes les fonctions indispensables et uniquement celles-là. Commenter ces prototypes (rôle des fonctions, de leurs paramètres ...).

Question 13 [1 pt]. Indiquer les fonctions qu'il faudrait ajouter au module de listes étudié en cours (voir première partie de l'énoncé) pour pouvoir réaliser le module de *geobucket* (ne pas se soucier du fait que les listes du cours sont des listes de `double` et pas d'`int`).

Question 14 [1 pt]. Notons $f(n)$ le nombre de comparaisons d'éléments effectuées par l'algorithme suivant, dans le pire des cas. La Figure 2 montre que $f(n)$ est compris entre les deux courbes $f_{\text{inf}}(n) = n \log_2(n+1)$ et $f_{\text{sup}}(n) = n \log_2(n) + n$. En déduire un équivalent asymptotique de $f(n)$. Justifier en quelques mots.

begin
  Soient L_1, L_2, ..., L_n un ensemble de n listes ne contenant chacune qu'un seul élément
  T = un *geobucket* initialisé à vide
  for i variant de 1 à n do
    Fusionner L_i avec le *geobucket* T
  end do
  L = conversion en liste de T
end

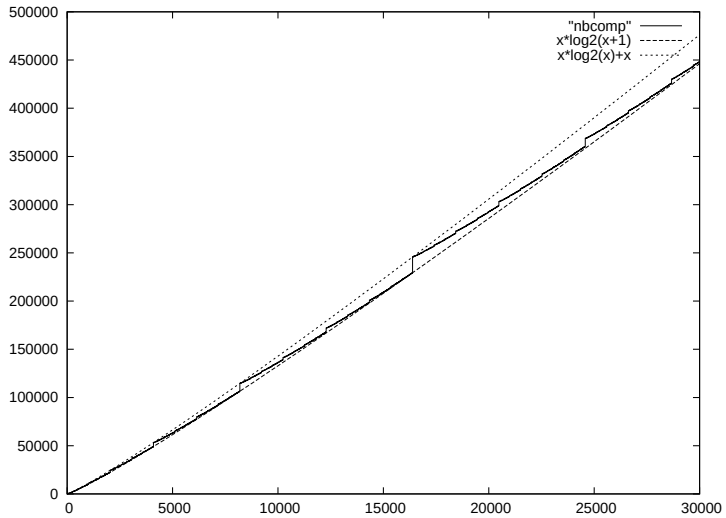


FIGURE 2 – La courbe expérimentale $f(n)$ est comprise entre les deux courbes $f_{\text{inf}}(n) = n \log_2(n+1)$ et $f_{\text{sup}}(n) = n \log_2(n) + n$.